Des situations connues aux traitements sur des données codifiées. Représentations mentales et processus d'acquisition dans les premiers apprentissages en informatiques

Jean-Baptiste Lagrange

▶ **To cite this version:**

Jean-Baptiste Lagrange. Des situations connues aux traitements sur des données codifiées. Représentations mentales et processus d'acquisition dans les premiers apprentissages en informatiques. Histoire et perspectives sur les mathématiques [math.HO]. Université Paris VII, 1991. Français. NNT : . tel-01270766

HAL Id: tel-01270766

https://theses.hal.science/tel-01270766

Submitted on 16 Feb 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

UNIVERSITE

PARIS 7

THESE pour le Diplôme

de DOCTORAT

Spécialité: Didactique des Disciplines

Présentée par: Jean-Baptiste LAGRANGE

SUJET de la THESE:

Des situations connues aux traitements sur des données codifiées.
Représentations mentales et processus d'acquisition dans les premiers apprentissages en informatique.

Soutenue le 10 décembre1991

JURY:

Jacques	ARSAC	Président
Janine	ROGALSKI	Directeur de la thèse
Charles	DUCHATEAU	Rapporteur
Jean-Michel	HOC	Rapporteur
Nicolas	BALACHEFF	Examinateur
Raymond	DURAND	Examinateur
Daniel	LACOMBE	Examinateur

Remerciements

Au moment de terminer cette thèse, je voudrais remercier les nombreuses personnes sans lesquelles elle n'aurait pu être menée à bien:

Jacques ARSAC et Charles DUCHATEAU sans le travail desquels il ne serait pas aujourd'hui possible de parler de Didactique de l'Informatique; je suis particulièrement reconnaissant à Jacques ARSAC d'avoir accepté de présider le jury, et à Charles DUCHATEAU d'avoir accepté la tâche de rapporteur,

Jean-Michel HOC dont les travaux en psychologie de la programmation m'ont été si utiles et qui a également accepté la tâche de rapporteur,

Janine ROGALSKI qui m'a dirigé et soutenu pendant les années de recherche et de rédaction,

Nicholas BALACHEFF et Raymond DURAND qui ont accepté de participer au jury,

mes collègues du CFIAP de CAEN, particulièrement François HEUZE et Françoise DEBART auxquels je dois d'avoir pu mener une recherche expérimentale, et de nombreuses discussions productives, les élèves de leurs classes qui ont accepté avec bonne humeur mes investigations,

et bien sûr Myriam, Mathieu et Clément.

A tous MERCI

Table des matières

Remerciements 1

Table des matières 3

Introduction 11

Partie I: Problématique

Chapitre 1 Les données en informatique

1. Préciser la notion de "structuration des données" 19
   - 1.1 La méthode choisie 20
   - 1.2 L'étude tente de répondre à quatre questions 20
   - 1.3 La sélection des ouvrages à étudier: 23
2. Un ouvrage "témoin": (D. KNUTH) 24
   - 2.1 Une présentation des structures de données 24
   - 2.2 L'histoire des structures de données 25
3. Une présentation décontextualisée (GAUDEL, SORIA, FROIDEVAUX) 26
   - 3.1 Un nouvel éclairage sur les structures de donnés 26
   - 3.2 La programmation abstraite 27
   - 3.3 L'élimination des représentations et ses conséquences 28
   - 3.4 Repérage d'un choix didactique 29
4. La création de programme comme "recontextualisation" (J.ARSAC) 31
   - 4.1 La programmation analytique 31
   - 4.2 Un exemple 32
   - 4.3 Essai de caractérisation de la démarche pédagogique 35
   - 4.4 Les capacités attendues de l'étudiant 36
5. Conclusions 36
   - 5.1 Qu'est ce que la structuration des données? 36
   - 5.2 La structuration des données, du point de vue du sujet 38
   - 5.3 L'évolution historique 39
   - 5.4 Les choix didactiques 40

Chapitre 2 Le sujet et l'activité de programmation

1. Les représentations mentales dans l'activité de programmation (HOC) 42
   - 1.1. Eléments théoriques 42
   - 1.2. Les degrés d'intériorisation du langage du dispositif 43
   - 1.3. Les S.R.T., les plans 44
2. La compréhension des écritures conditionnelles (T.R.G. GREEN) 45
   - 2.1 Les styles d'écritures conditionnelles. 46
   - 2.2 L'étude expérimentale 47
   - 2.3 Les modes de fonctionnement mental 47
   - 2.4 Conclusion 50
3. Le profit des "micro-mondes" (E. COHORS-FRESENBORG) 50
   - 3.1 Un dispositif entièrement descriptible 51
   - 3.2. Les micros-mondes 52

3.3. Stratégies de résolution et structures cognitives....52
3.4. Discussion....53
4. Une étude didactique (R. SAMURCAY)....54
4.1 Un problématique proche de celle de cette thèse....54
4.2 Les concepts de base....55
4.3 L'étude expérimentale....55
4.4 Les résultats....55
4.5 Discussion....56
5. Résumé des conclusions....57

Chapitre 3 Cadre général de recherche
1. Un public hétérogène sans finalité professionnelle....59
2. Le sujet confronté à l'activité de programmation....60
2.1 Les dispositifs informatiques....60
2.2 Les S.R.T. comme structures mentales locales....60
2.3 Le niveau des concepts informatiques....62
3. A partir de quels problèmes de programmation?....63
3.1 Les problèmes dans les premiers apprentissages....63
3.2 Données et traitements dans les premiers apprentissages....65
3.3 Des problèmes où les objets du monde réel et les données informatiques sont bien différenciés....68
3.4 Quelle utilisation des types prédéfinis ?....70
4. Conclusions du chapitre....72

Chapitre 4 Choix spécifiques pour l'expérimentation
1. Hypothèses et questionnement....73
1.1 Les difficultés des élèves, et la constitution des S.R.T....73
1.2 Des problèmes pour mettre à jour les difficultés et faire évoluer les S.R.T....74
1.3 Questions posées par la situation pédagogique: le cas du travail dirigé en présence de l'ordinateur....75
2. Le terrain: l'option informatique des lycées....76
2.1 Programmes et pédagogies de l'option informatique....76
2.3 Problèmes didactiques de l'option....77
2.4 Le cas des classes observées....77
3. Planification de la recherche....78
3.1 Observer de façon ponctuelle à des moments opportuns, des élèves ayant constitué, dans un cadre pédagogique donné, des ébauches de représentations....78
3.2 Tester la possibilité d'organiser la classe de problèmes que nous avons définie ci-dessus, en progression visant à faire évoluer les S.R.T. des élèves, et voir comment répondre aux questions concernant la situation pédagogique....79
4. Méthodologie....79
4.1 Les choix méthodologiques possibles....79
4.2 La méthodologie adoptée pour chacun des axes de l'étude expérimentale....80

Chapitre 5 Le débutant et le langage de programmation
1. Les structures fondamentales des langages. Affectation, notation

fonctionnelle, itération à un point de sortie....83
1.1 Les actions dans les langages impératifs....83
1.2 Les actions simples: l'affectation et les entrées/sorties....85
1.3 La notation fonctionnelle....87
1.4 Les actions complexes....88
2. Les chaînes de caractères....94
2.1 Définition de la structure....94
2.2 Les chaînes de caractères pour l'élève débutant....96
2.3 Travailler sur les chaînes avec des élèves débutants....99
3. Utilisation des langages de programmation réellement existants....100

Conclusion de la partie I

1. Les structures de données, les représentations mentales....103
2. Nos choix et hypothèses....104

Partie II: Représentations mentales et acquisition des chaînes de caractères

Chapitre 6 Etude des plans du S.R.T. relatifs aux objets

1. Classification des difficultés attendues d'élèves confrontés à une tâche impliquant l'emploi du type chaîne de caractères....112
1.1 La conception naïve du langage de programmation (Difficulté D.Langage)....112
1.2 Les difficultés propres à la structure de chaîne (Difficulté D.Sem.Cha.)....113
1.3 La conception des ordinaux, et la coordination ordinaux-cardinaux (Difficulté D.Ord.Card.)....114
1.4 Le typage des objets (Difficulté D.Type)....114
2. Apprentissages antérieurs des élèves concernés par l'épreuve....115
3. Présentation de l'épreuve EPR1....116
3.1 Les problèmes ONJOURB et RONJOUB....117
3.2 Le problème NOTES (question 3)....118
3.2 Le problème CHIFFRE (Question 5)....118
4. Exploitation des réponses des élèves....118
4.1 Les problèmes ONJOURB et RONJOUB....118
4.2 Le problème NOTES (Question 3)....119
4.3 Le problème CHIFFRE (Question 5)....120
4.4 Récapitulation par élève....120
4.5 Conclusions....120
5. Les entretiens....122
5.1 La conception naïve du langage de programmation....122
5.2 Les difficultés avec les ordinaux....124
5.3 L'installation d'un S.R.T. relatif au traitement des chaînes dans un S.R.T. opératoire plus général....126
6. Résumé du chapitre, et conclusions....127

Chapitre 7 Intervention des plans relatifs aux traitements

1. Hypothèses quant aux difficultés des élèves, et choix d'observation....131
1.1. Les difficultés propres aux actions complexes : intervention

des conditions, et des ruptures de séquentialité.....131
1.2 Nos choix d'observation.....132
2. Le contexte de l'épreuve.....135
2.1 Les élèves ont reçu un enseignement portant sur l'itération et sur l'analyse.....135
2.2 L'épreuve consiste en deux problèmes de "recherche-translation": CLASSE et CRYPTAGE.....137
3. Analyse des réponses au problème CLASSE.....141
3.1. Longueur de la sous-chaîne à laquelle comparer NOM$ 141
3.2. position à partir de laquelle rechercher la note comme sous-chaîne de CLASSE$.....142
3.3. Longueur de la sous-chaîne à extraire pour obtenir NOTE$ 143
4. Analyse des réponses au problème CRYPTAGE 144
4.1. Recherche dans la chaîne "azertyuiopqsdfghjklmwxcvbn" du caractère entré par l'utilisateur.....144
4.2. Expression du caractère résultat en fonction du caractère donné ou de sa position : classification des réponses.....145
4.3 Lien entre la structure algorithmique pensée par l'élève, et sa conception de l'ordinal, position du caractère donné.....146
4.4 Lien entre le mode de recherche de la lettre donnée dans le problème CRYPTAGE et la présence du compteur (I) dans le calcul de la position du premier caractère de la note dans le problème CLASSE.....149
4.5 Prise en compte du cas particulier constitué par la lettre "n" 149
4.6 Comparaison entre l'expression en langage naturel et le programme. 151
5. Confirmation de l'interaction Objets-Traitements (analyse d'un entretien).....152
5.1 Les conceptions des élèves 152
5.2 Le problème CRYPTAGE et le traitement du cas particulier 152
6. Conclusions 153

Chapitre 8 Etude quantitative

1. Vérifier la généralité des résultats du chapitre 6, opérer des comparaisons 155
2. Questions ouvertes et questions fermées.....156
3. Les conditions de passation et le texte de l'épreuve 157
3.1 Le problème CLASSE.....157
3.2 Le problème BONJROU 159
3.3 Le problème RUOJNOB 160
4. Dépouillement des réponses.....160
4.1 Les réponses au problème CLASSE 161
4.2 Les réponses au problème BONJROU 163
4.3 Les réponses au problème RUOJNOB.....165
5. Comparaison des performances des élèves selon la décision d'orientation du conseil de classe, et la classe dont ils sont issus.....166
5.1 Intérêt d'une classification des résultats selon la décision d'orientation.....166
5.2 Mise en évidence de différences entre classes.....167
5.3 Le choix d'un critère de réussite à l'épreuve.....167

5.4 Réussite à l'épreuve et décision d'orientation.....168
5.5 Réussite à l'épreuve et classe d'origine des élèves.....168
5.6 Cours et exercices sur les chaînes, pratiqués dans les différentes classes.....169
5.7 Relation entre la réussite des élèves et les activités en classe 170
6. Conclusions.....171

Conclusions de la partie II.....173

Partie III: Ingénierie didactique

Chapitre 9 Les problèmes et la situation pédagogique

1. Une classification des tâches sur les chaînes de caractères.....180
2. Principes de conception de séries de problèmes et de la situation de résolution.....181
2.1 Les énoncés mettent l'accent sur le calcul et l'affectation plutôt que sur l'affichage.....181
2.2 Le travail dans le langage de programmation présente des limites mais paraît le seul choix adapté à des débutants.....184
2.3 Les énoncés de problèmes peuvent être adaptés à différents langages.....188
2.4 L'articulation de l'itération et des types de données dans la progression choisie conditionne l'ordre dans lequel les énoncés sont abordés.....189
2.5 La structuration des données est généralement amorcée dans les énoncés.....189
2.6 Les problèmes portent sur des objets ayant une signification intuitive.....190
3 Présentation et analyse a priori des problèmes.....190
3.1 Première série (ALPHA): compréhension des arguments de la fonction sous-chaîne, problème simple de "recherche translation" 190
3.2 Seconde série (PIANO): Recherche-Translation avec arguments plus généraux.....192
3.3 Troisième série (TESTCHA): calcul sur des ordinaux, fonction longueur, conditionnelles à plusieurs variables.....192
3.4 Quatrième série (INSEE): type des données, fonctions de conversion de type.....194
3.5 Cinquième série (FILE): utilisation de la concaténation, coordination de structures algorithmiques complexes.....197

Chapitre 10 Expérimentation

1. Présentation de la classe et de la progression suivie par le professeur.....200
1.1 Eléments de la progression.....200
1.2 Le langage de programmation.....202
2. Présentation des élèves dont le travail a fait l'objet d'une observation, et des conditions de cette observation.....202
3. Analyse des séances de résolution.....204
4. L'épreuve sur papier à l'issue des séries de problèmes.....208
5. Analyse des réponses des élèves à l'épreuve sur papier.....211

6. Récapitulation des difficultés rencontrées par les élèves 215
7. analyse par type de difficulté et conclusions.....217
7.1. La difficulté (D.Type).....217
7.2. Les difficultés (D.Langage).....219
7.3. Les difficultés (D.Sem.Cha.).....224
7.4. Les difficultés (D.Ord.Card.).....228
7.5. l'interaction entre représentations des objets, et représentation des traitements.....230
7.6. l'intervention de structures alternatives:.....235
8. La situation d'apprentissage.....236
8.1 L'utilisation par les élèves des formulations en langage naturel 236
8.2 Le rôle de l'environnement du problème et des effets en retour du dispositif.....237
8.3 L'intervention de l'enseignant auprès des élèves.....239

Conclusions de la partie III

1. L'évolution des élèves 241
2. Des représentations avec lesquelles les régles du dispositif sont compatibles.....245
3. Retour critique sur l'analyse préalable.....245
4. La situation pédagogique: moyens de l'élève, interventions de l'enseignant.....246

Partie IV: Extension de la démarche de recherche aux booléens

Chapitre 11 Les objets de type booléen

1. Les booléens : contexte d'apparition d'une théorie des "booléens" 251
1.1 Le concept de booléen 251
1.2 Les définitions et propriétés de l'Algèbre de BOOLE 252
1.3 La place des booléens dans l'informatique et dans l'apprentissage de la programmation.....254
2. Les booléens dans les langages de programmation 258
2.1 Le cas des langages où les booléens n'existent pas comme type spécifique 259
2.2 Les langages où les booléens constituent un type à part entière 260
2.3 Le choix d'un langage de programmation.....261
3. Des hypothèses sur l'emploi du type booléen dans les premiers apprentissages en informatique.....261
3.1 L'apport de la résolution de problèmes impliquant les booléens 261
3.2 Les deux axes de problèmes avec leurs difficultés spécifiques 261
3.3 les conséquences du choix d'un langage de programmation.....262

Chapitre 12 Les booléens: étude expérimentale

1 Choix généraux.....263
1.1 Le type booléen est utilisé pleinement en classe de Première 263

1.2 Le choix d'une classe de Première restreint la généralité de l'observation 263
2 Les problèmes 264
2.1 INVITATION: un problème portant sur des "objets à deux états" 264
2.2 VILLAGE: un problème où les booléens codifient des relations entre objets du monde 267
2.3 Conditions de résolution 269
3 La recherche du premier problème (INVITATION) 269
3.1 La recherche sur papier 269
3.2 Le travail sur machine 272
3.3 Entretiens sur la résolution du premier problème (INVITATION) 274
3.4 Conclusion sur le premier problème (INVITATION) 275
4 La résolution du second problème (VILLAGE) 276
4.1 La recherche sur papier 276
4.2 Le travail sur ordinateur 277
4.3 Conclusions sur le second problème (VILLAGE) 279

Conclusions de la partie IV

1. Le choix du type des données 281
2. L'intégration différenciée des connecteurs logiques 282
3. La compréhension des expressions booléennes comme "conditions" 282
4. Comment les élèves peuvent-ils progresser dans l'emploi de booléens? 283

Conclusion générale

1. Les systèmes de représentation et de traitement 285
1.1 La construction de représentations mentales par le débutant 285
1.2 Dans certains problèmes, les réponses exactes peuvent être compatibles avec des représentations erronées 287
1.3 Représentations des objets et représentations des traitements 287
1.4 Evolution des représentations 288
1.5 Questions restant posées 289
2. Validité des problèmes choisis pour l'analyse des représentations et la construction d'une progression 290
3. La situation pédagogique 291
3.1 La prise en compte des représentations des élèves dans les interventions de l'enseignant et le rôle de l'erreur 291
3.2 Questions ouvertes 292
4. Les structures de données dans les apprentissages 292
4.1 L'interaction données-algorithmes et la construction de structures de données comme problème 292
4.2 L'institutionnalisation des fonctions construites en réponse à des problèmes 293
5. Directions de travail pour l'élaboration de stratégies pédagogiques 294
6. Pour conclure 295

BIBLIOGRAPHIE.....297

Introduction

Le travail présenté dans cette thèse concerne les apprentissages de la programmation par des débutants dans le cadre d'un enseignement d'informatique ouvert à tout public. Il a pour but de contribuer à clarifier les fondements de ce type d'apprentissage, de préciser et d'analyser certains aspects des difficultés qu'on y rencontre, et, à partir de cette analyse, de proposer et d'évaluer des directions de travail auprès des élèves.

Concernant l'étude de l'enseignement d'une discipline en situation scolaire, notre travail est donc une thèse de didactique. Aujourd'hui, à travers de multiples travaux de recherches, et à travers ses effets sur les pratiques d'enseignement, la didactique s'est imposée dans beaucoup de disciplines; cependant l'informatique semble constituer un cas à part où la nécessité et la conception d'un travail en didactique se discutent. En effet, si on peut faire remonter la naissance de l'informatique, en tant que domaine scientifique, à une cinquantaine d'année (en plaçant dans ce champ les travaux de Turing), la possibilité d'un enseignement à visée non strictement technique et professionnelle est apparue seulement dans les années 1970: par exemple, les formations d'enseignants destinés, dans les années 70 à utiliser l'informatique dans les établissements secondaires se faisaient chez les constructeurs, et beaucoup de licences et de maîtrises d'informatique (enseignements universitaires de second et troisième cycle) sont apparues au début des années 80.

Ces enseignements universitaires se sont développés de façon impétueuse. Par contre, les enseignements d'informatique ouverts à tout public, par exemple au niveau de débutants du secondaire, font l'objet de débats depuis le début des années 70, et il ne semble pas qu'un "régime stable" reposant sur un consensus ait été atteint. En France, l'option informatique des lycées, que nous présentons au chapitre 4, semblait pouvoir réaliser un certain point d'équilibre: elle a bénéficié d'un fort investissement de l'institution scolaire et des enseignants concernés, et d'un succès certain auprès des élèves. Malgré cela, elle a fait l'objet récemment d'une remise en cause vigoureuse, sans que les arguments échangés de part et d'autre nous aient paru avoir beaucoup évolué depuis le début des années 1970; son avenir est incertain[1].

Dans d'autres ordres d'enseignement (primaire, technique), l'enseignement de l'informatique se distingue mal des usages pédagogiques de l'informatique. L'idée même d'un enseignement de l'informatique à un public large, conçu comme un apport au développement des capacités conceptuelles, au même titre que par exemple, les mathématiques et la physique, semble donc être encore aujourd'hui incertaine, et aucune tradition d'enseignement affirmée ne peut aider à cerner mieux cette idée. En parallèle, la "didactique de l'informatique", au moins dans l'espace francophone, semble se développer principalement dans des directions qui ignorent ce type d'enseignement, et les problématiques concernent surtout les contenus à enseigner, plus que la façon dont les étudiants, en particulier les débutants pourraient accéder à ces contenus[2].

---

[1] Sur la création et le développement de l'option informatique des lycées, voir [BARON 88] (BARON G.L. *L'informatique comme discipline scolaire.* PUF). L'option Informatique des lycées s'est vue remise en cause sur la base d'un rapport du Conseil National des Programmes : (*Quel lycée pour demain* éditions CNDP-Le Livre de Poche)[M.E.N. 91]. Avec de nombreux enseignants et responsables de l'option, nous considérons que le jugement porté par ce rapport sur l'informatique au lycée (pages 135 à 141) ne tient aucun compte de la richesse des apports didactiques de dix années d'enseignement de l'option informatique. De façon tout aussi surprenante, le rapport propose «*un nouvel enseignement de l'informatique au lycée*» sans donner aucun contenu conceptuel à cet enseignement. Les dernières propositions ministérielles (Septembre 91) prévoient maintenant des "ateliers de pratique informatique", à l'initiative des établissements;

[2] Voir à ce propos [DUCHATEAU 91] : Charles DUCHATEAU "Synthèse d'un colloque en trois temps.." in Actes du Second Colloque Francophone de Didactique de l'Informatique. Une explication de la centration des réflexions sur les contenus à enseigner serait la rapide évolution des concepts de l'informatique; c'est certainement en partie vrai en ce qui concerne les enseignements

Nous pensons quant à nous, qu'un enseignement de l'informatique à un public large est nécessaire, du fait des multiples implications culturelles de cette discipline. En particulier, les disciplines traditionnellement enseignées, dans leurs développements récents, font référence aux concepts de l'informatique: on peut donc s'attendre à des difficultés si les élèves n'ont pas une initiation à ces concepts dans un enseignement spécifique. Pour nous, cet enseignement de l'informatique à un public large n'a pas su trouver sa place actuellement en grande partie à cause du trop petit nombre d'études réellement didactiques (c'est-à-dire analysant de façon liée les contenus conceptuels à enseigner, les modes d'appropriation de ces contenus par les élèves et les effets de l'enseignement). Il nous semble que de telles études seraient également profitables au développement d'enseignements plus spécifiques, universitaires ou professionnels, en permettant d'articuler les débats didactiques sur les contenus à enseigner, rendus nécessaires par l'évolution de la discipline, avec des connaissances sur les modes d'appropriation de ces contenus par les étudiants; en particulier, le développement de ces études didactiques permettraient, dans des enseignements à des publics spécifiques, de mieux cerner les caractéristiques de ces publics, et éventuellement de les élargir. Voilà pourquoi nous considérons l'étude de l'accès de débutants "tout public" aux concepts de base de l'informatique à la fois comme un point de passage obligé pour la didactique de l'informatique, et comme un champ de recherche largement ouvert (nous développons ce point au chapitre **3**). Notre thèse s'inscrit dans ce champ.

Ayant inscrit notre thèse dans le champ des premiers apprentissages en informatique, nous sommes parti de quelques hypothèses générales que nous avons ensuite précisées dans la première partie de cette thèse. Il nous a semblé d'une part qu'un certain consensus se dégage autour des notions ou concepts à enseigner, et de la manière de les aborder avec les élèves, et d'autre part, que des difficultés sont rencontrées par les élèves ou étudiants débutants, et que la nature et les raisons de ces difficultés paraissent résistantes à l'analyse.

Les contenus à enseigner comprennent la notion de variable, les types de base et les premiers traitements: dans un langage impératif on considère l'affectation, les traitements alternatifs, l'itération; dans un langage fonctionnel, on considère les définitions de fonction, les définitions alternatives et récursives. Il est généralement admis que ces notions ne prennent leur sens que dans leur interaction, et que, pour mettre en oeuvre cette interaction, l'apprentissage doit s'appuyer sur la résolution de "problèmes de programmation", c'est-à-dire la programmation de tâches pour un dispositif informatique (nous développerons ce choix au chapitre **3**). Il est également admis que les notions ou concepts doivent être présentés de façon suffisamment indépendante du dispositif et du langage utilisés pour que les concepts soient dégagés des particularités du langage et du dispositif, et que des méthodes doivent être données pour éviter une activité de programmation empirique, qui ne permettraient pas aux concepts de se dégager. Le programme de l'option informatique des lycées, constitue un exemple de contenus et de méthodes d'enseignement représentatif de ce consensus: nous le présenterons au chapitre **4**, et au chapitre **10**, nous discuterons, à la lumière des chapitre précédents, les méthodes qu'il propose.

Les difficultés des débutants ont pu être masquées au départ par le caractère informel de nombreux enseignements et elles continuent à être ignorées par certains auteurs[3]. Pourtant, l'extension de l'option informatique a mis en évidence leur

---

universitaires. Cependant, l'effet des choix fondamentaux concernant les concepts enseignés (en particulier le type de langage: impératif, fonctionnel, logique...) paraît difficile à apprécier de façon réellement objective actuellement. Nous montrerons que, par contre, la conception des problèmes supports de l'activité de résolution par les élèves, les choix concernant la mise en place des situations pédagogiques, constituent un ensemble de variables didactiques d'une grande richesse à partir duquel peut se faire un réel travail didactique.

[3]La persistance d'opinions sommaires sur les acquisitions en informatique se manifeste par exemple dans le rapport du Conseil National des Programmes cité dans la note n°1 ([M.E.N. 91]):*«Ceux qui sont fortement motivés et/ou très doués ne mettront pas longtemps à devenir des virtuoses du DOS et à en remontrer à leurs enseignants, si ce n'est à bien des informaticiens»*

caractère assez général (nous développerons ceci au chapitre 4); des études didactiques (que nous présenterons aux chapitre 2 et au chapitre 5) ont commencé une analyse de ces difficultés. Pour nous, il est clair que la prise en compte et l'analyse de ces difficultés est un point de passage obligé; en effet ce sont ces difficultés, dans la mesure où elles concernent bien les rapports du sujet avec les concepts informatiques, qui justifient et orientent l'enseignement. Ces difficultés semblent souvent inattendues pour l'enseignant, malgré la connaissance qu'il peut avoir du contenu qu'il enseigne et de son public: on manque de situations d'observation et de moyens d'analyse pour les caractériser, et orienter l'enseignement en conséquence. Une ambition de cette thèse est par conséquent de proposer de nouveaux "angles d'attaque" pour l'analyse de ces difficultés.

Nous avons pu faire une première hypothèse suite à une observation que nous avions menée dans le cadre de notre rapport de D.E.A.: nous avions étudié la réussite d'un groupe d'élèves-professeurs programmant en LOGO[4] dans la réalisation et la compréhension de procédures récursives, d'une part sur des objets graphiques en géométrie tortue, et d'autre part pour des algorithmes de calcul numérique (sommations...); nous avions mis en relation la réussite des élèves-professeurs dans chacun de ces domaines de programmation avec leurs résultats d'une part en géométrie et d'autre part en analyse. Il avait été possible de montrer une relation entre leurs productions dans un des deux domaines de programmation et leurs capacités dans le domaine mathématique correspondant (par exemple entre leurs procédures graphiques et leurs résultats en géométrie) alors que leurs capacités dans les deux domaines de programmation paraissaient peu corrélées: leur connaissance (acquise en mathématiques) du domaine concerné semblait influencer directement leur production en programmation, alors qu'il y avait peu de réinvestissement des capacités en programmation d'un domaine à l'autre.

Cette observation faite sur un public un peu particulier nous semble confirmée par des observations rapportées dans des publications récentes. [SAMURCAY ROUCHIER 90], suite à une étude expérimentale sur des élèves travaillant à l'écriture de procédures récursives, d'abord dans un environnement graphique, puis dans un environnement constitué de listes numériques, notent que «*le modèle relationnel qui a une valeur opératoire dans un cadre graphique, présente une grande fragilité. Dans un domaine nouveau (suites d'entiers) sa mise en oeuvre rencontre de difficultés importantes.*». [DAGDILELIS BALACHEFF CAPPONI 91] étudient la construction par des élèves d'un programme calculant la somme des n premiers nombres d'une part dans le tableur **Multiplan** et d'autre part avec le langage PASCAL; dans les deux cas ils notent des conduites influencées par les capacités «*anthropomorphiques*» que les élèves prêtent au dispositif, mais ces conduites différent notablement d'un contexte de programmation à l'autre, et en aucun cas les élèves ne réinvestissent dans la programmation PASCAL des démarches conduites pour la programmation sous **Multiplan**. Les schèmes de résolution mis en oeuvre par les débutants semblent donc très dépendants des objets sur lesquels portent le programme, des caractéristiques propres du langage, est donc plus généralement du contexte dans lequel est posé le problème.

Nous avons donc pensé que les difficultés rencontrées par des débutants pourraient s'analyser à partir de la distance entre le niveau auquel l'enseignement pense l'activité de résolution et celui auquel le sujet mène cette activité: l'enseignement pense l'activité de résolution comme la reconnaissance et la mise en oeuvre de concepts, c'est-à-dire d'entités suffisamment générales pour que leurs propriétés soient considérées de façon indépendante du contexte de programmation dans lesquels elles interviennent: par exemple, le schéma itératif de la multiplication de deux entiers comme addition itérée est considéré comme équivalent à celui du "retournement" d'une chaîne de caractères puisque, dans les deux cas, le schéma se caractérise par l'intervention d'un compteur parcourant un ensemble donné de valeurs et d'une variable d'accumulation. Par contre, les observations rapportées ci-dessus

[4] Le langage LOGO est présenté en annexe 5. On trouvera également au chapitre 3 une discussion des stratégies pédagogiques autour de la notion de "programmation en géométrie-tortue".

permettent de penser que les débutants raisonnent, quant à eux, au niveau du **domaine de tâches** que constitue le contexte du problème: leu compréhension des éléments du langage qu'ils utilisent pour la programmation est influencée par la compréhension qu'ils ont de ce domaine de tâche; ces éléments du langage peuvent donc difficilement être considérés séparément du contexte, et séparément les uns des autres, ce qui empêche que leur nature conceptuelle se dégage de l'activité de résolution, et qu'il y ait des transferts d'un domaine de tâche à l'autre: pour reprendre l'exemple ci-dessus, dans ce fonctionnement du débutant, le "retournement" d'une chaîne de caractères et la multiplication de deux entiers constituent pour des débutants en informatique, deux domaines séparés de façon étanche où les concepts ne peuvent avoir de signification commune.

Cette hypothèse a pour conséquence d'obliger à "repenser le problème d'informatique" dans les premiers apprentissages. En effet, pour que des débutants puissent dégager, à partir de leur activité de résolution, la spécificité des objets informatiques en tant qu'entités abstraites connues par leurs seules règles de fonctionnement, et le caractère formel du traitement de ces objets par le dispositif informatique, il paraît nécessaire qu'ils rencontrent dans l'activité de programmation, les limites des schèmes de résolution qu'ils déduisent de leur connaissance intuitive du domaine de tâche, quand ceux-ci se révèlent non pertinents pour le traitement informatique, ou non compatibles avec les règles qui régissent les entités informatiques qu'ils utilisent. Il faut donc que les situations proposées aient un sens intuitif pour le sujet, tout en imposant des différenciations entre le niveau des objets et traitements intuitifs, et le niveau des objets et traitements informatiques (ce point sera développé au chapitre **3**).

Une première conséquence est la nécessité que les énoncés de problèmes fassent intervenir une véritable **codification** des données, c'est-à-dire une correspondance entre les objets intuitifs du problème (données, résultats) et les entités informatiques abstraites sur lesquelles portent le traitement. Nous sommes donc amenés à faire intervenir dans ces problèmes non seulement des structures algorithmiques, mais aussi ce qu'en informatique on appelle la **structuration des données**, c'est-à-dire la recherche, par un processus d'abstraction, de propriétés des données suffisamment dissociées du contexte pour être compatibles avec les éléments du langage (types de données, structures algorithmiques) du dispositif.

C'est pourquoi le premier chapitre de cette thèse tente de préciser les notions liées à la structuration des données, la façon dont elles s'articulent avec les structures algorithmiques dans les problèmes, la façon dont elles peuvent être prises en compte dans des stratégies pédagogiques. S'appuyant sur l'analyse d'ouvrages universitaires, cette analyse est menée sur un domaine conceptuel qui dépasse celui des premiers apprentissages. Il nous a semblé en effet important de repérer des axes conceptuels et des choix didactiques valables aux différents niveaux auxquels l'informatique est enseignée, de façon à ne pas s'enfermer dans une conception de l'informatique et de son enseignement spécifiques aux débutants. Le premier chapitre permet donc de décrire des axes généraux concernant les concepts à enseigner et les choix didactiques. Ayant décrit ce cadre général, il nous est plus facile de dégager des spécificités d'un enseignement à des débutants.

Pour préciser notre hypothèse générale, nous avons également besoin d'outils d'analyse pour étudier les conduites du sujet (et particulièrement du sujet débutant) en situation de résolution. C'est l'objet du chapitre **2**: nous tirons parti d'une part de recherches en psychologie de la programmation s'appuyant sur l'observation de programmeurs en situation de résolution, ([HOC 77] et [GREEN 77]), et d'autre part d'études didactiques menées autour de la mise en place d'enseignements d'informatique à des débutants ([SAMURCAY 85] et [COHORS-FRESENBORG 87]). Ce chapitre nous permet en particulier de dégager la notion de **Système de Représentation et de Traitement (S.R.T.)** comme structure mentale mise en oeuvre par le programmeur au cours de son activité de résolution, et la façon dont différents systèmes s'articulent chez le sujet à différents niveaux de compétence. Il nous permet également d'examiner comment les modes de constitution de ces systèmes peuvent varier selon le contexte dans lequel s'effectue l'activité de résolution.

Le chapitre 3 prend en compte les conclusions des chapitres 1 et 2 pour préciser plusieurs points de cette introduction: choix du public à observer, conduites du sujet dans l'activité de résolution, conception du problème de programmation. Au chapitre 4, nous exposons les hypothèses spécifiques à cette thèse: hypothèses sur les **difficultés des élèves** et la **constitution des S.R.T.**, hypothèses sur les **problèmes** susceptibles de mettre à jour les difficultés et de faire évoluer ces Systèmes, questions posées par la **situation pédagogique**, particulièrement celle du travail dirigé en présence de l'ordinateur. Nous présentons également la **méthodologie** adoptée pour l'étude expérimentale visant à valider ces hypothèses: terrain expérimental, planification de la recherche, méthodologie.

Les choix relatifs aux problèmes nous conduisent à situer l'étude expérimentale dans le cadre de l'utilisation d'un langage de la classe des langages impératif; en effet, la structuration des données à l'aide des listes, qui est cohérente avec les structures algorithmiques en langage fonctionnel ou en programmation logique nous paraît imposer un mode de raisonnement d'emblée du niveau du raisonnement par récurrence, alors que nous pensons que, par exemple, l'itération sur une chaîne de caractères peut, dans un premier temps s'interpréter comme une répétition d'action, plus accessible aux élèves que nous considérons; par ailleurs, le choix d'un langage impératif, le plus répandu en option informatique des lycées nous simplifie la recherche d'un terrain d'observation.

Ce choix étant fait, les concepts de base que nous considérons sont l'affectation et les écritures fonctionnelles, ainsi que l'itération.à un point de sortie; nous les étudions au chapitre 5 et nous faisons des hypothèses sur leur utilisation par les élèves de façon à disposer d'un cadre d'analyse pour, dans un premier temps, préparer les problèmes sur lesquels nous projetons de faire travailler les élèves, et, après ce travail, classifier les difficultés qu'ils auront effectivement rencontrées. Dans la même perspective, nous présentons la structure de chaîne de caractères qui constitue le premier type dont nous considérons l'utilisation par des élèves pour des codifications.

La partie II (chapitres 6, 7 et 8) rapporte une série d'observations ponctuelles en classe de seconde option informatique, menée à l'aide d'épreuves et d'entretiens portant sur la résolution de problèmes impliquant des codifications à l'aide de chaînes de caractères. Les observations des chapitres 6 et 7 sont de nature clinique. Elles permettent de compléter le cadre d'analyse et de tirer des premières conclusions: le chapitre 6 examine au sein des **S.R.T.**, les représentations mentales des objets codifiés sous forme de chaînes de caractères, en lien avec la compréhension de l'affectation et des écritures fonctionnelles; le chapitre 7 prend en compte l'interaction, dans les **S.R.T.**, de ces représentations des objets d'une part, et de schémas de résolution relatifs aux traitements d'autre part, à travers la résolution d'une classe de problèmes faisant intervenir l'itération à un point de sortie et la structure de chaîne. L'observation du chapitre 8 porte sur plusieurs classes et sur davantage d'élèves; son but est de vérifier le caractère de généralité des résultats obtenus aux chapitres 6 et 7.

La partie III examine la possibilité d'organiser la classe de problèmes que nous avons définie au chapitre 3, en *progression* visant à faire évoluer les **S.R.T.** des élèves, et tente de répondre aux questions concernant la situation pédagogique. Elle consiste en un travail d'**ingénierie didactique**, et une élucidation d'une situation pédagogique de résolution: la situation de travail dirigé par petit groupe en présence de l'ordinateur (chapitre 9). Au chapitre 10, nous rapportons une observation portant sur 4 élèves particulièrement en difficulté, à partir de la progression et de la situation définies au chapitre précédent. Cette observation nous permet la mise en évidence de nouveaux aspects des difficultés étudiées dans les chapitres précédents, et une étude de l'évolution de ces difficultés; elle nous permet également un retour critique sur la progression de problèmes établie au chapitre précédent, et sur l'analyse qui la fonde, ainsi que de premières réponses aux questions concernant la situation pédagogique.

La partie IV est le pendant de la partie II concernant un autre type de données de base: les booléens. Au chapitre 11, nous étudions les booléens dans le même esprit que les chaînes de caractères au chapitre 5: pour disposer d'une première idée de la façon dont des débutants peuvent utiliser les booléens, de façon à construire une observation; par ailleurs nous déterminons deux directions de problèmes, impliquant des

codifications à l'aide de booléens. L'observation d'élèves d'une classe de première option informatique est rapportée au chapitre **12**: comme aux chapitres **6** et **7**, nous l'avons menée à partir à l'aide d'épreuves et d'entretiens portant sur la résolution de problèmes. Elle constitue une étude clinique, moins approfondie que celle des parties **II** et **III**, mais montrant que la problématique que nous avons développée dans ces parties ne se limite pas au type "chaînes de caractères".

Sur le plan pratique, à la fin de chaque partie, une conclusion tente de faire le point sur l'avancement des idées. A la fin de la thèse, une conclusion reprend les résultats obtenus et les perspectives ouvertes. Elle est suivie de la bibliographie. A partir du chapitre **5**, un certain nombre de textes d'analyse, d'énoncés de problèmes, de protocoles d'observation et de productions d'élèves est renvoyé en annexe; nous avons fait ce choix pour ne pas alourdir les chapitres concernés et permettre la consultation simultanée de ces annexes et de la thèse. Un second volume comprend donc les annexes, numérotées comme les chapitres correspondants, de l'annexe **5** à l'annexe **12**.

Partie I: Problématique

Chapitre 1: Les données en informatique
Chapitre 2: Le sujet et l'activité de programmation
Chapitre 3: Cadre général de recherche
Chapitre 4: Choix spécifiques pour l'expérimentation
Chapitre 5: Le débutant et le langage de programmation

Chapitre 1 Les données en informatique

aspects "unificateurs"
nécessités d'un enseignement à des débutants
évolution des conceptions
choix didactiques

1. Le but de ce chapitre: préciser la notion de "structuration des données"

Nous avons indiqué en introduction que les choix généraux de cette thèse nous conduisent à étudier les conceptions des élèves et leur évolution au cours de la résolution de problèmes où la codification des données et la nécessité d'un traitement comme calcul sur ces données codifiées font rencontrer au programmeur débutant les spécificités du dispositif informatique. Les éléments théoriques de psychologie de la programmation dûs à J.M. HOC [HOC 77][5] que nous présenterons au chapitre suivant, nous permettrons d'apercevoir que l'enjeu de cette activité de résolution est la constitution progressive chez l'élève de *Systèmes de Représentation et de Traitement* (**S.R.T.**), c'est-à-dire d'un ensemble de représentations mentales concernant d'une part les objets traités, et d'autre part les traitements eux-mêmes, et la création par l'élève de codes permettant le passage entre différents **S.R.T.**, en particulier un (des) **S.R.T.** déduit(s) directement de son expérience sensible, et un (des) **S.R.T.** adapté(s) à la représentation des données et au traitement par un dispositif informatique.

Les travaux en psychologie ou didactique de l'informatique (particulièrement ceux que nous présenterons au chapitre suivant: [COHORS-FRESENBORG 87][6], [GREEN 77][7], [SAMURCAY 85][8] et ceux que nous présenterons en annexe au chapitre 4: [LABORDE, BALACHEFF, MEIJAS 1985][9] et [ROGALSKI 84][10]), considèrent seulement les éléments algorithmiques du langage du dispositif, et s'intéressent donc essentiellement aux conceptions relatives au traitements au sein du **S.R.T.** Quant à nous, notre but est d'approfondir l'aspect "Données" (ou "Objets") des **S.R.T.**, et d'examiner les articulations avec l'aspect "Traitement" (ou "Algorithme"). Voyons dans un exemple, comment une tâche de réalisation d'un traitement pour un dispositif informatique implique la différenciation et la coordination des deux aspects: le problème est, un entier étant donné, d'isoler le chiffre des unités de son écriture en base 10. Dans le contexte intuitif (par exemple pour un opérateur humain), la tâche est immédiate ; au contraire, s'il s'agit de réaliser un programme pour un dispositif informatique, il faudra se préoccuper de la façon dont le nombre entier est représenté dans le dispositif : si le nombre est représenté par la "chaîne" de ses chiffres, il

---

[5] HOC J.M. (1977) «Role of mental representation in learning a programming language» *Int. Man-Machine Studies* n°9

[6] ECOHORS-FRESENBORG E. (1987) «The Benefit of Microwords in learning computer programming» *Osnabrücker Schriften zur Mathematik* 04/87

[7] GREEN T.R.G. (1977)«Conditional program statements and their comprehensibility to professionnal programmers» *J.occup.Psychol* n°50

[8] SAMURCAY R. (1985) «Signification et fonctionnement du concept de variable informatique chez des élèves débutants» *Educational Studies in Mathematics* n°16 Kluwer Academic Publishers

[9] LABORDE C., BALACHEFF N;, MEIJAS B. (1985) «Problématique et genèse du concept d'itération, une approche expérimentale» *ENFANCE* 2-3

[10] ROGALSKI J. (1984) *Sur une séquence didactique en informatique* ; rapport de recherche IMAG n°50

faudra utiliser ou construire une fonction permettant d'isoler le dernier élément d'une chaîne ; s'il est représenté par une entité numérique, seules les opérations algébriques pourront lui être appliquées et dans ce cas, l'algorithme de résolution consistera en la recherche du reste dans la division entière par 10. Dans le premier cas, le nombre est une donnée de type "chaîne"; dans le second cas, c'est une donnée de type numérique. A chacun des types correspond un traitement (algorithme) spécifique.

L'exemple précédent nous montre que, la codification des données, et le traitement considéré comme calcul sur des données codifiées font intervenir la "*structuration des données*" en tant que constituant du langage du dispositif. Cette notion de "structuration des données". est très présente en informatique, aussi bien dans l'enseignement, à partir d'un certain niveau, que dans les pratiques professionnelles, alors que les cursus au niveau de l'initiation (par exemple le programme officiel de la classe de Seconde, option informatique que nous présenterons au chapitre 4) ou le curriculum élaboré par E.COHORS-FRESENBORG, ([COHORS-FRESENBORG 87] présenté au chapitre 2 § 3.1) mettent l'accent sur les structures algorithmiques. Nous nous proposons donc, dans ce chapitre, de tenter de préciser la notion de structuration des données et ses rapports avec les algorithmes (correspondant à l'aspect "Traitement"). Nous examinons en premier lieu les contenus informatiques constituant cette notion, mais aussi les choix didactiques qui peuvent être faits dans l'enseignement de cette notion, et nous faisons des hypothèses sur la façon dont cette notion intervient dans les premiers apprentissages en informatique.

1.1 La méthode choisie: une étude d'ouvrages d'enseignement de niveau universitaire

[CHEVALLARD 85][11] a désigné par «*transposition didactique*» les mécanismes qui, un «*savoir savant*» étant donné, conduisent à l'élaboration d'un «*savoir enseigné*». Il a montré, à propos des mathématiques, la complexité de ces mécanismes, et la fréquence des phénomènes de «*substitution d'objet*», qui peuvent conduire à des objets d'enseignement très éloignés, voire sans rapport avec les objets de savoir. Concernant l'informatique, la «*transposition didactique*» des concepts liés à la structuration des données nous semble très peu avancée, et donc, si l'on veut éviter les constructions artificielles qui résultent de substitutions d'objets, il importe de se référer au savoir constitué : c'est pourquoi nous tentons dans ce chapitre de préciser la notion de **structuration des données**, à partir de l'étude de textes de référence et d'ouvrages d'enseignement de niveau universitaire[12]. Bien entendu, ce savoir constitué est le produit d'une élaboration au long de la jeune histoire de l'informatique et des divergences apparaissent entre les auteurs, concernant la présentation et/ou l'utilisation des structures de données ; nous nous efforcerons, à travers le choix des textes, et l'étude que nous en ferons, de faire apparaître cette élaboration et ces divergences, et d'en examiner les conséquences pour l'enseignement.

1.2 L'étude tente de répondre à quatre questions

Nous avons indiqué que le but de chapitre est de préciser la notion de structuration des données, dans la perspective d'une étude des difficultés des élèves débutants. Ce but peut être poursuivi à travers la recherche de réponses aux quatre questions suivantes :

---

[11] CHEVALLARD Y. (1985) *La transposition didactique* La pensée sauvage

[12] Un savoir constitué a l'avantage pour nous d'une certaine stabilité dans le temps; des constructions conceptuelles naissent en permanence dans le domaine de l'informatique, et nous pensons prudent d'attendre qu'elles aient trouvé une forme définitive avant de tenter une "transposition didactique"; c'est pourquoi nous n'abordons pas ici la "programmation orientée objets" bien qu'elle soit concernée de façon évidente par les questions que nous posons sur la structuration des données. On pourra en savoir plus sur les langages de la "programmation orientée objets" en consultant: [MILLOT 88] Introduction aux langages orientés objets *Bulletin de l'association E.P.I. n°52*

question 1 : les aspects "unificateurs"

Existe t'il des aspects "unificateurs", qui permettraient que, même vues à un niveau élémentaire par des débutants, les notions liées à l'organisation des données s'inscrivent dans une continuité conceptuelle avec les structures de données (Files, Piles, Arbres...) de l'informatique ?

Pour illustrer cette question, prenons un exemple dans l'enseignement des mathématiques : le concept d'équation[13] intervient aussi bien dans l'enseignement de la soustraction des naturels au cours préparatoire que dans l'enseignement universitaire par exemple pour présenter l'inversion des matrices ; il donne un sens à des opérations telles que la soustraction, la division, la recherche de primitives, et permet que les mécanismes opératoires soient placés à leur juste place, dépendante du problème posé. Si par contre, le concept d'équation n'est pas présent dans l'apprentissage d'une opération, comme par exemple la division, ou la recherche de primitives, les mécanismes opératoires de recherche d'un quotient ou les régles de calcul des primitives se constituent en objets d'enseignement autonomes, et on assiste à ce que CHEVALLARD désigne sous le nom de «*substitution d'objet*»[14].

De la même façon, il nous semble que les premiers traitements demandés aux élèves, par exemple sur les chaînes de caractères auront davantage de sens si l'on peut leur trouver une continuité conceptuelle avec les structures de données employées dans l'informatique professionnelle, et enseignées au niveau universitaire: prenant leur sens dans cette continuité, ces premiers traitements donneront alors de meilleures possibilités d'accès à des structures et langages de niveau supérieur, voire à l'utilisation avertie de progiciels.

question 2 : les nécessités d'un enseignement à des débutants

A côté de ces aspects "unificateurs", y a t'il des nécessités propres à un enseignement destiné à des débutants ? Pour reprendre l'exemple ci-dessus, on sait que l'acquisition de la soustraction des naturels au cours préparatoire ne se réduit pas à la question de l'existence et de l'unicité d'une solution à l'équation `a + x = b` ; elle nécessite par exemple de s'intéresser à la façon dont les élèves du cours préparatoire peuvent se représenter et manipuler cette équation.

Compte-tenu de la méthode employée dans ce chapitre (étude de textes de

---

[13] En fait, il s'agit d'un concept sur des concepts préexistants : les variables et l'égalité, donc un "méta-concept".

[14] La division des entiers, par exemple, est fondée sur l'équation `a = b q + r` (`a` et `b` étant donnés, trouver `q` et `r` tels que `r < b`.Le mécanisme opératoire, est la façon concrète de poser l'opération (barre horizontale et verticale, disposition des nombres, calculs élémentaires...). Dans un apprentissage qui vise l'acquisition de la notion de division, les éléments du mécanisme opératoire, en particulier les calculs élémentaires s'appuient sur cette équation. Par contre, le mécanisme opératoire présenté de façon isolée, constitue un savoir faire d'utilité pratique, qui se *substitue* à l'objet mathématique "division" : un élève peut posséder parfaitement cet algorithme, et être incapable de donner un sens aux résultats obtenus. De même, la notion de primitive est fondée par la recherche, une fonction `g` étant donnée d'une fonction `f` dont la dérivée soit `g` (l'équation est donc `f' = g`, `f` étant l'inconnue) . On a certains théorèmes d'existence, et d'unicité (à une constante près), mais pas d'algorithme de résolution. En fait, la recherche de primitives s'organise de façon plus ou moins sophistiquée autour d'une démarche inductive (se ramener à des fonctions dont on sait qu'elles sont les dérivées de fonctions connues). De même que pour la division, l'équation `f' = g` peut fonder cette démarche inductive. Dans ce cas, la recherche de primitives participe bien de l'acquisition de la notion de primitive. Si, au contraire, l'enseignement se réduit à l'acquisition de certains schémas de recherche de primitive, là aussi, un savoir faire d'utilité pratique se *substitue* à l'objet mathématique. On pourrait multiplier les exemples : soustraction, inversion des matrices... En même temps, ce méta-concept d'équation rend possible une articulation dans le temps, des acquisitions : un élève ayant compris que la soustraction est la résolution d'un problème sera mieux préparé à la division On a donc un domaine conceptuel, où la notion d'équation (recherche de solutions à une relation donnée) permet que les apprentissages soient réellement centrés sur les objets qu'ils visent, et en même temps, les met en perspective.

référence), il est à prévoir que ces nécessités apparaîtront en fait "en négatif", c'est à dire qu'il faudra les rechercher comme des éléments absents, ou considérés comme accessoires dans les textes étudiés : nous verrons que tous les ouvrages de référence étudiés précisent qu'ils s'adressent à un public "déjà initié" ou ayant déjà "un peu pratiqué". Nous rechercherons donc ce qui peut différencier un "déjà initié" d'un non-initié, un sujet qui a "un peu pratiqué" d'un sujet qui n'a pas pratiqué du tout.

question 3 : la structuration des données dans l'évolution de l'informatique

Comment la notion de structuration des données apparaît-elle au long de la jeune histoire de l'informatique ? Quel éclairage cette genèse historique donne t'elle à la notion ?

Nous ne voulons pas, en posant cette question, signifier que les conditions d'apparition des concepts doivent guider de façon mécanique la façon dont ces concepts sont abordés avec des élèves débutants. Mais il nous semble que la connaissance de ces conditions permet une meilleure compréhension de la signification des concepts et permet de mieux fonder des choix pédagogiques. Nous prendrons, pour illustrer cette question, l'exemple du débat bien connu entre les styles de programmation, qui a des implications directes y compris sur l'initiation, puisque le choix d'un style de programmation détermine souvent le choix du langage de programmation. On oppose ainsi la programmation "*impérative*" (caractérisée par l'emploi de l'affectation et de l'itération pour l'évolution des variables) à la programmation "*fonctionnelle*" (caractérisée par la création d'une nouvelle instanciation des variables à chaque étape du traitement, et la récursivité) (voir [PAIR 88][15], [CLOUTIER 88][16]). Le point essentiel de l'historique est que la programmation fonctionnelle dispose dès le départ de l'informatique (avant même la construction des ordinateurs) d'une théorisation mathématique achevée. Elle est ainsi plus ancienne que la programmation impérative, alors que, paradoxalement, la disponibilité plus récente dans le public de langages fonctionnels lui donne une apparence de nouveauté. La programmation impérative a dû attendre une vingtaine d'année avant que des théoriciens lui donnent des bases mathématiques. E.DIJKSTRA (préface à [DIJKSTRA 76][17]) note que, selon lui, de nombreux informaticiens de formation mathématique marquent une préférence pour la programmation fonctionnelle parce qu'ils se sentent plus à l'aise dans un domaine bénéficiant d'une théorisation plus ancienne, et donc plus classique, et qu'ils tendent pour cela à considérer les formes récursives comme plus naturelles, ce qu'il conteste fortement.

question 4 : choix didactiques

Quels choix de présentation et/ou d'utilisation des structures de données repère t'on dans des ouvrages de référence ? Ces choix sont-ils à mettre en relation avec des choix didactiques pour un enseignement destiné à des débutants ? Il semblera peut-être paradoxal de tenter de repérer dans des ouvrages de niveau universitaire, des choix didactiques ayant une validité pour un enseignement destiné à des débutants. Nous soulignons d'abord qu'il n'existe pas de tradition d'un enseignement de l'informatique destiné à des débutants qui permettrait un repérage de choix didactiques spécifiques à ce niveau, surtout dans le domaine qui nous intéresse, et que d'autre part, des choix didactiques sont d'autant plus intéressants qu'ils se révèlent pertinents pour des cursus de niveaux variés.

---

[15] PAIR C. (1988) «L'apprentissage de la programmation» *Actes du premier colloque francophone de didactique de l'informatique* Editions EPI

[16] CLOUTIER J.F. (1988) «Apports de différents paradigmes de programmation comme autant d'outils de pensée» *Actes du premier colloque francophone de didactique de l'informatique* Editions EPI

[17] DIJKSTRA E.(1976) *A discipline of programming* Prentice Hall

1.3 La sélection des ouvrages à étudier: un ouvrage "témoin" et deux manuels représentatifs de choix actuels

Pour repérer l'élaboration de la notion de structuration des données au cours du développement de l'informatique, il nous semble utile de disposer d'un ouvrage classique, témoin de l'histoire de l'informatique, avant d'aborder l'étude de textes actuels. En tant qu'ouvrage classique, «*The art of computer programming*» [KNUTH 67][18] présente l'avantage d'être contemporain de la constitution de l'informatique comme domaine scientifique : commencé en 1962, dans la perspective d'un seul volume en 12 chapitres, l'ouvrage compte finalement 7 (gros) volumes, couvrant chacun un semestre universitaire. Nous étudions le chapitre 2 (volume 1) qui comporte une description détaillée de diverses "structures de données" et, une note historique précisant les conditions d'apparition des différentes structures, et l'évolution de la notion.

Concernant les ouvrages actuels, le choix est vaste, parmi les ouvrages d'enseignement et/ou de référence traitant des structures de données. En voici une sélection parmi les ouvrages français ou "classiques" traduits en français :

- «*Algorithms + Data Structures = Programs*» [WIRTH 76][19] commence par un chapitre sur les structures de données fondamentales (non dynamiques). N.Wirth note dans la préface que la structure algorithmique d'un programme , et la structuration de ses données sont inséparables, mais que de façon intuitive, la donnée précède l'algorithme : «*il faut disposer d'objets, avant d'accomplir des opérations sur ces objets*».
- «*Méthodes de programmation*» [MEYER BAUDOIN 78][20] est un ouvrage de référence qui présente les structures de données après une introduction aux langages, et les premiers éléments algorithmiques (structures de contrôle, sous-programmes).
- «*Algorithmique et représentation des données*» [LUCAS 83][21] présente une progression ordonnée suivant des structures de données de complexité croissante : file, puis arbre, puis graphe ; à chaque structure de donnée est associée un ensemble de traitements algorithmiques.
- «*Les bases de la programmation*» [ARSAC 83][22] : on connaît l'apport de l'auteur dans l'émergence de l'informatique comme discipline scientifique, sa contribution à l'enseignement de cette discipline au niveau universitaire, et à la constitution d'un cursus au niveau du secondaire. En particulier, Jacques ARSAC a fortement contribué à la mise en place de l'option informatique des lycées, et au suivi pédagogique de cet enseignement (qui constitue notre terrain d'étude et que nous présenterons au chapitre **5**). Par rapport à d'autres traités, l'ouvrage de J. ARSAC a l'originalité de ne pas présenter une progression basée sur les structures de données ; l'objectif de l'ouvrage est de donner des méthodes de construction d'algorithmes, et la structuration des données n'apparaît que pour les besoins de ces méthodes.
- «*Programmation avancée*» [BOUSSARD MAHL 85] [23]comporte deux parties, la première étant consacrée aux algorithmes, la seconde aux structures de données ; pour chaque structure de données, un ensemble de possibilités de représentation est donné et discuté.
- «*Types de données et algorithmes*» [GAUDEL SORIA FROIDEVAUX 86][24] systématise la structuration des données par la notion de «*type abstrait*», et

---

[18]KNUTH D. (1967) *The art of computer programming Tome 1* Addison-Wesley

[19]WIRTH N. (1976) *Algorithms + Data Structures = Programs* Prentice Hall

[20]MEYER B. BAUDOIN C. (1978) *Méthodes de programmation* Eyrolles

[21]LUCAS M. (1983) *Algorithmique et représentation des données Tome 2* MASSON

[22]ARSAC J. (1983) *Les bases de la programmation* DUNOD

[23]BOUSSARD J.C. MAHL R. (1985) *Programmation avancée* Eyrolles

[24]GAUDEL M.C. SORIA M. FROIDEVAUX Ch. (1986] *Types de données et algorithmes* INRIA

introduit à la «*programmation abstraite*».

- «*Construire les algorithmes*» [PAIR MOHR SCHOTT 88][25] est un ouvrage où la structuration des données intervient directement dans les méthodes d'élaboration d'algorithmes ; selon les auteurs, «*le type des objets en jeu peut guider la construction de la solution*», par exemple, le type du résultat peut permettre d'obtenir une décomposition du problème. Cet ouvrage nous a paru tout à fait intéressant pour la façon dont les liens entre structuration des données et structures algorithmiques sont utilisés pour la construction de programmes.

Parmi ces ouvrages, nous avons choisi d'étudier plus à fond et de présenter dans cette thèse, tout d'abord [GAUDEL SORIA FROIDEVAUX 86], puis [ARSAC 83]. Tout d'abord, le premier ouvrage nous permet de présenter les notions de type abstrait et de représentation sous une forme décontextualisée[26]. Ensuite, les deux ouvrages ont pour nous l'avantage de présenter des options très divergentes ; en effet le premier se présente comme un catalogue de structures, et privilégie dans l'activité de programmation l'utilisation des données sous forme de type abstrait. Comme nous le verrons, J. ARSAC émet des réserves sur ce type d'exposé, en ce sens qu'il considère que cela peut être un obstacle à la mise en oeuvre de méthodes de construction de programmes. Chacun des trois sous-chapitres suivants (numérotés **2, 3, 4**) est consacré à l'étude d'un ouvrage. Le sous-chapitre **5** tente de répondre aux questions posées en **1.2.**

2. Un ouvrage "témoin": «*The art of computer programming*» t.1 (D. KNUTH)

Nous avons indiqué en introduction que nous projetons d'étudier cet ouvrage à titre de témoin de la constitution de l'informatique comme domaine scientifique. C'est pourquoi nous rapporterons et analyserons dans la seconde section de ce paragraphe les informations historiques données par D. KNUTH à la fin du chapitre sur les structures de données. Mais il est également possible et utile d'étudier cet ouvrage comme un exposé scientifique et didactique et d'analyser en particulier comment l'auteur présente les structures de données ; c'est ce que nous faisons dans le premier paragraphe de ce sous-chapitre.

2.1 Une présentation des structures de données qui lie l'aspect fonctionnel et les représentations pour une machine virtuelle

L'introduction aux structures de données est constituée par le second chapitre, sous le titre «*Information Structures*». Les structures de données sont présentées comme un moyen de relier entre elles des données, d'un double point de vue :

- une structure de données peut être définie par sa «*représentation*»[27] dans un langage (ici un assembleur virtuel, le langage `MIX`), c'est-à-dire par un procédé effectif permettant de relier des données utilisant les éléments du langage : structuration en cellules ("`nodes`"), comportant un nombre fixé de mots mémoires,

---

[25] PAIR C. MOHR R. SCHOTT R. (1988) *Construire les algorithmes* Dunod

[26] Nous empruntons les termes "décontextualisé", et "recontextualisé" pour caractériser des formes de présentation d'un savoir à G.BROUSSEAU ("Fondements et méthodes de la didactique des Mathématiques" Revue Recherche en didactique des Mathématiques 7.2 1986) [BROUSSEAU 86]. Pour G.BROUSSEAU, la communication d'un savoir nouveau par le mathématicien qui l'a crée suppose que ce mathématicien ait débarrassé ce savoir de «toutes les réflexions inutiles, (de) la trace des erreurs commises et des cheminements erratiques». G.BROUSSEAU note qu'ainsi, «le producteur du savoir dépersonnalise, décontextualise et détemporalise le plus possible ses résultats» de façon à produire le texte d'un savoir «aussi objectif que possible».
Nous dirons qu'un savoir de ce type est présenté sous une forme"*décontextualisée*".

[27] Le terme *représentation* est ici employé dans son acception en informatique. Dans la suite, nous préciserons "*représentations mentales*" pour les représentations du sujet (constituantes de ses **S.R.T.**) à chaque fois que ce sera nécessaire pour la clarté de l'exposé.

divisées en champs ("fields") dont certains peuvent avoir pour valeur des liens ou pointeurs, ("links"), c'est-à-dire une adresse permettant l'accès à une autre cellule.

- La structure de donnée peut être vue également comme l'ensemble des opérations (ou fonctions) possibles sur les données, et leurs propriétés : «*the design of computer representations depends on the desired function of the data as well as on its intrinsic properties*». C'est l'aspect "*fonctionnel*".

La présentation des structures de données joue en fait sur les deux points de vue, d'une manière que nous rapportons de façon schématique ci-dessous, à propos des listes linéaires :

- étape 1. Les listes linéaires sont d'abord présentées comme formées de cellules à plusieurs champs, un des champs constituant un lien permettant d'accéder à la cellule suivante ; D.KNUTH utilise donc pour définir cette structure générale, une *représentation* dans la machine virtuelle qu'il a défini .
- étape 2. Les fonctions possibles sur une telle structure sont énoncées, et des listes linéaires particulières (piles, files) sont distinguées par des restrictions sur les fonctions (par exemple, la pile dispose seulement de l'insertion et de l'effacement à une extrémité de la liste) ; D. KNUTH utilise donc cette fois une *définition fonctionnelle* pour particulariser ces listes linéaires.
- étape 3. On revient au point de vue de la *représentation* pour la machine virtuelle en examinant la réalisation effective en mémoire de chaque structure particulière : la représentation sous forme contiguë des piles est décrite, ainsi que les difficultés qu'elle pose ; puis les représentations à l'aide de liens simples ou doubles sont décrites et discutées.

Ce va-et-vient entre les deux points de vue (fonctionnel et représentation en machine) nous a semblé remarquable ; il s'accompagne de représentations figurées, par exemple des réseaux de chemin de fer pour illustrer les structures de pile et de queue. Dans la conclusion du chapitre, D. KNUTH indique qu'il a insisté sur les liens entre les structures de données et leurs représentations en machine «*pour des raisons pédagogiques*». L'ouvrage que nous étudierons au sous-chapitre suivant (sous-chapitre 3) base, au contraire sa présentation sur la séparation entre la définition algébrique et les représentations.

2.2 L'histoire des structures de données

La partie «*history and bibliography*» du livre de D.KNUTH donne des indications précieuses sur le contexte dans lequel les structures de données ont été introduites :

- les listes linéaires et les tableaux de données contiguës en mémoire datent des premiers ordinateurs. D. KNUTH note qu'une des motivations ayant conduit à la réalisation de machines à programme enregistré (c'est à dire conçues selon l'architecture de VON NEUMANN) est la possibilité que le programme modifie lui-même un opérande dans une de ses propres instructions, cette particularité étant utilisée pour parcourir les files ou les tableaux avant l'apparition de registres d'index.
- Les notions de pile (dernier entré, premier sorti) et de queue (premier entré, premier sorti)[28] proviennent du domaine de la comptabilité, où elles interviennent dans la gestion de stocks. TURING a utilisé dès 1947 une pile pour la conservation des adresses de retour des sous-programmes, et l'utilisation de piles contiguës en mémoire a été courante dès les premiers temps de l'informatique.
- Les techniques d'allocation de la mémoire pour des listes linéaires de longueur variable apparaissent plus tard, notamment en 1963 pour l'écriture d'un compilateur COBOL.
- Les techniques d'utilisation de liens (ou pointeurs) pour la structuration des données sont nées avec les recherches de NEWELL, SHAW, SIMON sur la résolution heuristique de problèmes, et la réalisation du système G.P.S.[29]. D. KNUTH note que

---

[28] On emploie aussi les acronymes LIFO (Last In First Out) et FIFO (First In First Out)

[29] Sur les capacités et la réalisation du système G.P.S. (General Problem Solving) voir G.ERNST

ces techniques ont inspiré principalement les personnes concernées par la simulation du raisonnement humain, avant d'être reconnues comme des outils de base pour la programmation, et d'être étudiées pour elle-mêmes.

- concernant les chaînes de caractères, D. KNUTH indique l'existence de plusieurs systèmes traitant cette structure (COMIT et SNOBOL que nous présenterons en annexe au chapitre 5) ; ces systèmes sont basés sur les mêmes algorithmes que les structures étudiées dans le chapitre, mais sont conçus de façon à ce que l'utilisateur n'ait pas à se préoccuper de la façon exacte dont l'ordinateur procède.

A travers ces indications de D. KNUTH, nous assistons à la naissance de la notion de structure de données : les premières structures sont frustes, simplement dérivées de l'organisation matérielle de la mémoire ; par différence, la notion d'algorithme est, à l'époque où l'informatique commence vraiment (avec les premiers ordinateurs) une notion mathématique déjà ancienne, qui a été développée et formalisée avec les travaux de A.TURING, (par exemple, la notion de programme interpréteur est présente dans ces travaux)[30].Les structures de données sont par contre introduites à l'occasion de problèmes concrets rencontrés dans la programmation d'applications informatiques, et peuvent être inspirées de domaines extérieurs à l'informatique. La mise en oeuvre concrète d'une structure de données pour une application précède souvent la publication d'études à son sujet ; les structures de données apparaissent d'abord comme des techniques limitées à certains problèmes, avant d'être considérées comme des outils généraux, puis comme des sujets d'études. Selon D. KNUTH la théorisation des structures de données date de la fin des années 60.

3. Une présentation décontextualisée: «*Types de données et algorithmes*» (M.C. GAUDEL, M. SORIA, Ch. FROIDEVAUX)

Nous avons indiqué en introduction à ce chapitre, que nous projetons, par l'étude de cet ouvrage, d'une part de prendre connaissance d'une définition systématisée des structures de données, d'autre part d'examiner un point de vue qui donne la priorité à l'emploi des types abstraits en programmation. Comme dans les autres ouvrages, nous tenterons également de mettre en évidence les choix pédagogiques des auteurs, et leurs conséquences. Les auteurs déclarent s'adresser à un public «*déjà familiarisé avec la programmation*» et donnent comme but à leur ouvrage la présentation des types de données et des algorithmes fondamentaux, «*dont la connaissance est indispensable à tout informaticien, en développant de façon accessible les résultats récents dans ce domaine*».

3.1 Un nouvel éclairage sur les structures de donnés : types abstraits et représentations

Comme dans l'ouvrage précédent (D.KNUTH) les structures de données sont envisagées sous deux aspects:

- le «*type abstrait*», c'est à dire «*une notation pour (...) décrire (les données) ainsi que l'ensemble des opérations qu'on peut leur appliquer et les propriétés de ces opérations*».
- la «*représentation* des données» : «*représentation concrète du type de données en terme d'objets du langage de programmation utilisé, ainsi que des procédures ou des fonctions correspondant aux opérations du type*».

De façon plus précise, un type abstrait constitue une spécification des données d'un problème ; cette spécification se compose d'ensembles de valeurs («*sortes*»), de noms d'opérations et de leurs «*profils*» (c'est à dire la donnée de leurs ensembles de

---

A. NEWELL GPS: *A case study in generality and problem solving* Academcic Press 1969.

[30]On peut trouver une présentation accessible de ces travaux de A.TURING dans J..WEIZENBAUM "Computer Power and Human Reason" traduit en Français sous le titre de "Puissance de l'Ordinateur et Raisonnement de l'Homme" Editions d'Informatique 1981.

départ et d'arrivée, ces ensembles étant définis à l'aide des «*sortes*») et des propriétés de ces opérations données sous la forme d'un ensemble d'axiomes. Le type abstrait peut laisser non définies les opérations concernant certaines sortes, et les propriétés de ces opérations, soit que les types correspondant à ces sortes soient déjà définis, soit que leur définition n'importe pas à ce niveau (on dit qu'on «*utilise*» ces types).

Dans [KNUTH 67] la représentation des données, en servant à réaliser de façon concrète une structure pour une machine donnée, se situait dans le domaine de la mise en oeuvre. Dans la présentation de [GAUDEL SORIA FROIDEVAUX 86], la représentation a une dimension supplémentaire: elle constitue un lien (abstrait) entre structures de donnée. Examinons par exemple page 80, la représentation des arbres généraux (c'est à dire des arbres ayant un nombre quelconque de fils à chaque noeud). Parmi d'autres solutions, la *représentation* à l'aide d'arbres binaires est proposée. Elle est fondée sur le résultat suivant : il existe une correspondance bijective entre les arbres généraux ayant `n+1` noeuds et les arbres binaires ayant `n` noeuds. En plus de son utilité pratique, le résultat a un intérêt théorique puisqu'il permet de déduire des théorèmes concernant les arbres généraux de théorèmes sur les arbres binaires (notamment dans le domaine de la complexité des algorithmes).

Par rapport à la présentation de D. KNUTH, l'aspect "fonctionnel" de la structure se précise donc par la notion de «*type abstrait*» qui est en fait un système formel, avec un alphabet de symboles, des régles de formation des expressions et un système d'axiomes. L'aspect "représentation" s'enrichit, en ce sens que la représentation n'est plus seulement l'implémentation d'une structure pour une machine donnée dans le cadre de l'activité concrète de programmation, mais un lien abstrait entre deux structures considérées comme des systèmes formels.

3.2 Types abstraits et représentations dans l'activité de programmation.La programmation abstraite

Nous avons vu au paragraphe précédent que les notions de type abstrait et de représentation constituent des catégories générales non nécessairement liées à la programmation comme activité concrète. En revenant à l'activité de programmation, la séparation entre la spécification des données sous forme de types abstraits, et leurs représentations permet selon les auteurs, une démarche de programmation «*descendante*». Dans cette démarche descendante, la première version d'un algorithme doit pouvoir utiliser les données indépendamment de leur implémentation en machine ; à ce niveau, ce sont les opérations sur les données, et leurs propriétés qui importent : on programme donc sur des données définies comme des types abstraits. C'est seulement ensuite qu'on se posera la question des différentes possibilités de représenter les types de données, compte-tenu de l'environnement de programmation : on choisira les types de données correspondant aux types abstraits, s'il en existe dans le langage de programmation, sinon, on construira des représentations à l'aide de types plus généraux, pour chacun desquels on cherchera ensuite une représentation... Selon les auteurs, «*Les avantages de cette approche sont multiples : la conception est plus simple, puisqu'on n'a pas à prendre en compte des détails de programmation ; elle est faite une fois pour toutes, quelle que soit la représentation concrète choisie ultérieurement pour le type abstrait*». Les auteurs indiquent qu'une pratique courante consiste au contraire à travailler au départ sur une représentation des données du problème dans le langage de programmation : il s'agit donc d'une démarche «*ascendante*». Mais même dans cette démarche, si elle est correctement menée, la représentation concrète n'intervient que comme support intuitif : on manipule les données seulement par les opérations du type ainsi défini. «*On programme en fait avec des types abstraits*».[31]

[31] Les termes "*descendant*" et "*ascendant*" font référence à la conception des algorithmes. Dans une conception descendante, le problème à résoudre est décomposé en sous-problèmes ; chaque sous-problème est explicité (on dit aussi spécifié) ; si cette explicitation conduit à un algorithme de résolution du sous-problème, on écrit le programme correspondant, sinon on itére la décomposition au sous-problème Dans une conception ascendante, au contraire, le programmeur

3.3 L'élimination des représentations et ses conséquences

Dans ce type de programmation, appelée *programmation abstraite*, la définition des données comme type abstrait peut être considérée comme une systématisation de la démarche de "spécification" par laquelle l'informaticien précise les propriétés des objets dans le domaine qu'il doit modéliser. Elle constitue bien-sûr un progrès par rapport à une démarche consistant à représenter d'emblée les objets du domaine par les types pré-définis d'un langage.

Mais il nous semble que la conception de la programmation présentée par les auteurs constitue une forme particulière de la programmation abstraite, donnant un statut secondaire à la représentation; en effet, dans l'ouvrage, le raisonnement sur les données considère seulement les types abstraits, leur représentation apparaissant seulement comme une nécessité pratique sans signification particulière ; dans les exemples de résolution de problèmes présentés par les auteurs, elle est souvent laissée en exercice. Il nous semble que cette forme "systématisée" de la programmation abstraite introduit des *restrictions* aux possibilités d'évolution d'un programme. Nous examinerons au § 4, avec la présentation de [ARSAC 83], les conséquences de ces restrictions. L'exemple suivant va montrer quelles sont ces restrictions: supposons qu'un programme doive opérer sur des **nombres complexes.** Le programmeur pourra fort bien effectuer l'analyse du problème et rédiger un programme comme si le langage disposait de façon primitive du type correspondant (opérations et propriétés du corps des nombres complexes, avec les limitations propres aux données informatiques). Pour réaliser ensuite l'implémentation concrète dans un langage disposant seulement du type "réel", il remplacera simplement chaque variable complexe c par deux variables réelles x et y. Au moment de coder par exemple, une addition de nombres complexes, il remplacera l'affectation $c_3 \leftarrow c_1 + c_2$, par les deux affectations: $x_3 \leftarrow x_1 + x_2$ et $y_3 \leftarrow y_1 + y_2$ (ce sera une "régle de traduction" purement syntaxique). Si, de plus, le langage dispose de la possibilité de définir un type "couple de réels" et d'écrire des fonctions ayant comme argument(s) et comme résultat ce type, il pourra écrire un jeu de fonction, et coder ensuite son programme comme si le type "complexe" était primitif dans le langage. Les deux démarches sont conformes à l'esprit de la programmation abstraite. Au contraire, supposons que le programmeur s'autorise à utiliser les variables internes à la structure, *indépendamment d'une opération du type*: dans l'exemple, il utilisera x ou y en dehors des opérations sur les nombres complexes et passera, de ce fait, d'un problème sur N données complexes à un problème sur $2N$ données réelles. Il se trouve que, parfois, cette façon de changer de problème peut apporter un éclairage nouveau à l'algorithme, et permettre la création d'un nouvel algorithme (nous donnerons un exemple au §**4.2.1**). Par contre, ce comportement, dans le cadre d'une programmation modulaire peut conduire à des difficultés: si un module considère le problème comme portant sur $2N$ nombres réels, et un autre module sur N réels positifs et N réels de l'intervalle $[0, 2\pi]$ (ce qui correspond à une représentation polaire), ils auront des difficultés à communiquer.

C'est pourquoi, dans la programmation professionnelle, est apparue la nécessité de langages où l'on peut définir des types de données non primitifs tels que les éléments internes de leur représentation ne soient pas accessibles au programmeur les utilisant, le type étant donc connu de ce programmeur par ses seules spécifications (abstraites). C'est le cas du langage **ADA**; une structure de données peut être utilisée par un programmeur à partir d'un module (*package*) où seule la spécification de cette structure est connue, la représentation étant privée (*private*), c'est à dire que les objets et les procédures utilisées pour cette représentation ne sont pas connues du programmeur utilisateur, et doivent pouvoir varier d'une version à l'autre du module sans que les

---

écrit des parties d'algorithmes répondant à des problèmes partiels, avant d'envisager l'organisation générale de son programme. Nous verrons au chapitre suivant, avec la lecture de [HOC 77], que la conception d'un programme de façon descendante suppose que le sujet dispose de représentations mentales opératoires lui permettant d'envisager directement une «*solution abrégée*» directement compatible avec le dispositif.

programmes l'utilisant aient à être modifiés. Par exemple, un programmeur utilisant un module "nombres complexes" ne saura pas si, de façon interne, les nombres complexes sont les couples de réels d'une représentation cartésienne ou les couples formés d'un réel positif et d'un angle orienté d'une représentation polaire[32]. La programmation abstraite, si elle conduit à éliminer la possibilité de raisonner sur des représentations introduit une restriction à l'utilisation des structures de données, un peu comme l'élimination du GOTO au profit d'instructions conditionnelles structurées introduit une restriction dans le domaine des traitements. Dans les deux cas, ces restrictions se justifient par des raisons "conceptuelles", l'analyse en terme de type abstrait et de programmation structurée permettant de raisonner indépendamment des détails d'implémentation. Ces raisons conceptuelles rencontrent des nécessités pratiques dans le cas de projets importants, où la tâche est décomposée en modules dont la programmation est répartie entre les membres d'une équipe. Nous nous proposons de montrer au paragraphe 4, que cette élimination des représentations des données peut poser problème quant il s'agit d'enseigner la construction de programmes.

3.4 Repérage d'un choix didactique de présentation «*décontextualisée*» des notions liés aux objets

Nous avons indiqué au début de ce chapitre qu'un des buts que nous poursuivons à travers l'étude d'ouvrages de référence est la mise en évidence de choix de présentation des notions et des stratégies pédagogiques liés à ces choix. La section D (étude d'un exemple) de [GAUDEL SORIA FROIDEVAUX 86] est présentée comme une «*analyse d'algorithme*» ; elle est pour nous l'occasion d'examiner comment les auteurs utilisent les relations entre structure algorithmique et structures de données, et comment ils conçoive un exposé des notions destiné à des étudiants.

Etude d'un exemple proposé dans l'ouvrage.

Le problème est le suivant : «**recherche des deux plus grands éléments d'une liste** L **de** n **entiers**». Les auteurs s'intéressent d'abord au sous-problème consistant à rechercher le plus grand élément. L'algorithme proposé pour ce sous-problème est bien connu : le premier élément de la liste est affecté à une variable M, puis chacun des éléments suivants est comparé à M, et , dans le cas où il est supérieur, sa valeur est affectée à M ; à chaque pas, M a ainsi pour valeur le plus grand élément de l'ensemble des éléments déjà parcourus. Il est démontré que l'algorithme est optimal en nombre de comparaisons, c'est à dire que tout algorithme résolvant le problème effectuera au moins autant de comparaisons que l'algorithme décrit ci-dessus (c'est-à-dire n-1).

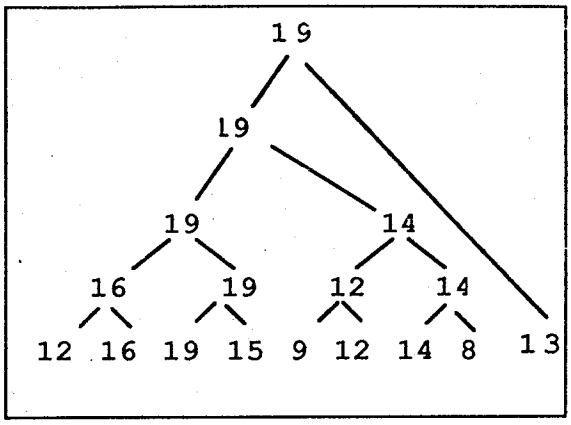
On revient ensuite au problème complet ; les auteurs donnent un premier algorithme, que nous appelerons *algorithme de sélection ordinaire*, et qui consiste à reprendre l'algorithme de recherche du plus grand élément, énoncé ci-dessus, puis à appliquer le même algorithme à la liste privée du plus grand élément. La complexité totale en comparaisons (recherche du plus grand élément + recherche du second) est (n-1)+(n-2) c'est-à-dire 2n-3.

Les auteurs exposent ensuite un second algorithme que nous appelerons *algorithme de sélection par tournoi* où le sous-problème de recherche du plus grand élément se fait par comparaisons deux à deux, de façon à obtenir comme résultat annexe une organisation des données suivant un arbre binaire dont les n feuilles (noeuds terminaux) sont les éléments de la liste et dont chacun des n-1 noeuds restant est le "vainqueur du tournoi" entre son fils droit et son fils gauche (c'est à dire le plus grand des deux).

---

[32] Cet exemple est tiré de : J.G.P. BARNES An overview of Ada, in software practice & experience, vol. 10 1980 [BARNES 80].

Voici par exemple l'arbre obtenu à partir de la liste : 12 16 19 15 9 12 14 8 13



La recherche du *second plus grand élément* se fait ensuite sur le sous-ensemble `E` des éléments qui ont été opposés au maximum dans une comparaison ; en effet, il fait partie de cet ensemble, car il est nécessairement vainqueur dans toutes les comparaisons auxquelles il a participé, sauf celle qui l'a opposé au maximum. Le sous-ensemble `E` se parcourt en partant de la racine, et en choisissant à chaque étape le fils dont le noeud est le maximum ; les éléments de `E` sont obtenus comme noeuds de l'autre fils (dans l'exemple ci-dessus, les éléments de `E` sont, dans l'ordre, 13, 14, 16, 15)

Chacun des noeuds non terminaux de l'arbre correspond à une comparaison. La complexité en comparaisons de la partie "recherche du plus grand élément" est donc de `n-1`, comme dans le premier algorithme. Le nombre d'éléments de `E` est visiblement égal à la hauteur de l'arbre, c'est-à-dire à l'entier `h` tel que 2^h soit immédiatement supérieur à `n`. Le nombre de comparaisons nécessaires dans la seconde étape est égal à `h-1`, alors qu'il était de `n-2` dans le premier algorithme. Le second algorithme est donc meilleur du point de vue du nombre de comparaisons et la suite de l'étude consiste à montrer qu'il est optimal (au pire), c'est-à-dire que tout algorithme procédant par comparaisons a une complexité supérieure ou égale. Ce qui nous intéresse dans cet exemple, c'est l'apparition, d'une solution à l'autre, d'une structuration des données, paraissant mystérieuse car elle ne découle pas de la nature des données de l'énoncé. Nous allons, ci-dessous tenter d'examiner les raisons de cette apparition, et de caractériser la méthode d'exposition adoptée par les auteurs.

Discussion

Comparons les deux algorithmes rapportés ci-dessus du point de vue de la structuration des données. Pour l'algorithme de sélection ordinaire, une structure de file est suffisante ; par contre, l'algorithme de sélection par tournoi utilise un arbre binaire. Dans la section consacrée à la structure d'arbre, la nécessité de la structure d'arbre a été présentée comme découlant de la nature des données : «*tournoi de tennis; pedigree d'un cheval de course ; expressions arithmétiques*». On découvre dans ce exemple que la structuration des données est liée en fait à l'algorithme et non à la nature des données : dans l'exemple, la structure d'arbre est introduite pour l'obtention d'un algorithme efficace, et non parce que les données y conduisent "naturellement". Pour prendre une comparaison dans l'enseignement des mathématiques, on pourrait considérer un exposé des bases de l'algèbre linéaire où l'on introduirait les matrices sans référence aux systèmes d'équations ni aux applications linéaires, en essayant cependant de trouver des raisons "naturelles" à cette introduction. Puis on développerait les régles des opérations et les propriétés de ces opérations. Enfin on montrerait que l'outil matriciel s'applique de façon commode à des problèmes d'applications linéaires et aux systèmes d'équations. Dans le cas de cet exposé (potentiel) sur les matrices, comme dans le cas de la présentation des structures de données dans l'ouvrage que nous étudions, il y a contradiction entre les raisons invoquées pour l'introduction d'un concept nouveau et les raisons réelles, telles qu'elles apparaissent lorsqu'il est fait usage du concept. Cette contradiction nous paraît découler de la volonté de présenter les concepts nouveaux sous leur forme la plus systématique, ce que G. BROUSSEAU ([BROUSSEAU 86]) appelle une présentation

décontextualisée. (Cf. note 22 ci-dessus).

Pour montrer qu'il y a bien une possibilité de choix didactique, comparons l'analyse d'algorithme étudiée ci-dessus à la façon dont [PAIR MOHR SCHOTT 88] présentent ces mêmes algorithmes : l'algorithme de sélection ordinaire est discuté, et l'explication de sa lenteur est recherchée dans le fait **qu'on ne garde aucune mémoire des comparaisons effectuées** lors de l'étape de recherche du plus grand élément ; les auteurs montrent que la mémorisation des comparaisons **conduit à une organisation arborescente des données** et donc à l'algorithme de sélection par tournoi. Ici, la structure de données apparaît comme une nécessité pour la construction d'un algorithme efficace. L'arbre est resitué dans un contexte qui lui donne son sens (il est «*recontextualisé*»). Le calcul de la complexité de l'algorithme obtenu est l'occasion de démontrer un théorème sur les arbres binaires : l'étudiant est donc amené à s'intéresser aux propriétés des arbres binaires en dehors du contexte introduit par le problème (la notion est «*redécontextualisée*»)[33].

L'ouvrage que nous venons d'examiner nous a permis d'une part de situer la programmation abstraite comme un moyen de raisonner sur les propriétés des données du problème indépendamment de leur représentation; nous avons montré qu'un raisonnement portant exclusivement sur les types abstraits et renvoyant la représentation à une tâche de codage introduit des restrictions sur l'évolution des algorithmes, et mis en évidence une possibilité de choix didactique: nous avons opposé la présentation des auteurs où algorithme et structuration des données apparaissent séparés et figés, d'une présentation où ces deux composantes de la programmation prendraient leur sens par leur évolution conjointe.

4. La création de programme comme "recontextualisation": «*Les bases de la programmation*» (J.ARSAC 83)

«*Les bases de la programmation*» est un ouvrage d'enseignement de niveau universitaire. Nous avons indiqué dans l'introduction de ce chapitre que le choix de cet ouvrage est motivé tout d'abord par la contribution de l'auteur à l'enseignement de l'informatique au niveau universitaire, et à la constitution d'un cursus au niveau du secondaire, ensuite par les réserves que l'auteur exprime sur le mode d'utilisation des structures de données que nous avons tenté de caractériser à travers l'étude de l'ouvrage précédent. Il nous semble que l'explicitation des motifs de ces réserves peut être féconde pour le repérage de choix didactiques.

4.1 La programmation analytique

L'auteur développe dans cet ouvrage des méthodes de construction d'algorithmes regroupées sous le nom de «*programmation analytique*» : son ambition est de ne pas se contenter d'un exposé des algorithmes, mais de fournir à l'étudiant des **moyens effectifs de résolution** de problèmes de programmation. Nous présentons dans ce paragraphe les grandes lignes de la programmation analytique, et la place des structures de données dans ces méthodes; au paragraphe suivant nous illustrerons cette présentation par l'étude d'un exemple. Puis nous tenterons de caractériser la démarche pédagogique de l'ouvrage.

De façon générale, la méthode consiste, un procédé de calcul étant donné, à l'exprimer sous forme d'une fonction récursive, puis d'un programme se composant d'une initialisation et d'un sous-programme récursif avec variables globales, à passer ensuite à un programme itératif (il existe certains schémas de transformation qui

---

[33] Selon G.BROUSSEAU ([BROUSSEAU 86]), le professeur doit se livrer à un travail «*inverse*» de celui du savant : «*il doit produire une recontextualisation et une repersonnalisation des connaissances*». Les connaissances vont ainsi devenir «*la connaissance de l'élève, c'est-à-dire une réponse assez naturelle à des conditions (...)indispensables pour qu'elles aient un sens pour lui*». Le travail des élèves est de «*redécontextualiser et redépersonnaliser leur savoir, (...) de façon (à l') identifier avec le savoir qui a cours dans la communauté scientifique et culturelle de leur époque*».

permettent ce passage). Le programme itératif obtenu est le résultat de plusieurs transformations et est donc souvent lourd ; en examinant la signification des variables, on peut le simplifier pour obtenir un programme plus élégant, plus facile à implémenter et à valider.

Dans cette présentation résumée, la programmation analytique semble concerner au premier chef les algorithmes. Elle implique cependant *une conception des rapports entre structures de données et algorithmes.* La **structure de pile**, par exemple, est présentée en liaison étroite avec la **transformation des sous-programmes récursifs** (où elle sert à conserver les données nécessaires aux calculs en attente). Nous verrons dans l'exemple exposé au paragraphe suivant (§ 4.2) que la pile est d'abord utilisée de façon abstraite ; ses propriétés sont déterminées par l'usage qui en est fait pour la transformation du schéma récursif. Ensuite, J.ARSAC montre qu'il peut se trouver un choix de représentation de la pile, dans lequel les variables servant à cette représentation ont une signification dans le problème. Le programme itératif résultant est ainsi construit à l'aide du *choix d'une représentation de la structure de données*, ce qui ne serait évidemment pas possible en programmation abstraite (nous avons indiqué au § 3.3 de ce chapitre, les *restrictions* apportées par la programmation abstraite).

4.2 Un exemple d'évolution mutuelle de l'algorithme et de la structuration des données

Voici l'énoncé d'un exemple de problème tiré de l'ouvrage, où la structure de pile subit la transformation évoquée au paragraphe précédent; nous rapportons de façon abrégée la façon dont la construction d'algorithmes s'applique à cet exemple, de façon à illustrer le paragraphe précédent, puis nous commentons.

4.2.1 L'énoncé et les transformations du programme

On souhaite réaliser le produit d'un nombre x par un entier k, en utilisant seulement les opérations suivantes :

- multiplication d'un nombre par 2,
- addition de deux nombres,
- reste et quotient entier d'un entier par 2.

Le calcul s'appuie sur la définition récursive :

```
f(k,x) : SI k=1 ALORS x
                SINON SI pair(k) ALORS 2*f(k/2,x)
                                 SINON f((k-1)/2,x) +
                                       f((k+1)/2,x)
                      IS
         IS
```

• Transformation en procédure récursive:

L'application de la définition de l'énoncé n'est pas optimale : certains calculs sont effectués plusieurs fois : ainsi dans

`f(19,x) = f(9,x)+f(10,x)= f(4,x) + f(5,x) + 2* f(5,x)`

le calcul de f(5,x) est effectué 2 fois.

Une première série de transformations conduit à une procédure récursive à trois arguments avec un seul appel récursif : `k` est passé "par valeur", c'est-à-dire que, à l'appel de `F(k/2,u,v)`, une nouvelle instance de `k` est crée, pour la durée de l'exécution de la procédure, avec comme valeur, la valeur antérieure divisée par 2, et que, à l'issue de l'exécution, la valeur antérieure est restituée ; par contre **u** et **v** sont passées par référence, ce qui veut dire qu'il s'agit des mêmes variables d'un bout à l'autre du programme. les transformations qui conduisent à cette forme ne sont pas triviales, mais elles ne concernent pas directement notre propos.

```
F(k,u,v) : SI k=0 ALORS u←0 ; v←x
                  SINON F(k/2,u,v);
                        SI impair(k) ALORS u←u+v ;
                                           v←2*v
                                     SINON v←u+v;
```

```
                                                u←2*u
                                IS
            IS
```

l'hypothèse de récurrence[34] vérifiée à chaque sortie du sous-programme est : *[[u=k*x ; v=(k+1)*x]]* , ce qui permet de fixer les conditions initiales. Le programme calculant le produit de n et b est donc :

```
x←b ; F(n,u,v) ; resultat← u
```

• Elimination des paramètres:

La transformation suivante consiste à éliminer les paramètres dans F , et donc à utiliser à leur place des variables globales ; ceci ne pose pas problème pour u et v qui sont les mêmes variables d'un bout à l'autre du programme. Par contre, la valeur de k, antérieure à un appel de la procédure doit pouvoir être restituée à la sortie de cette procédure[35]. En fait, seule la parité de k sert dans la suite de l'exécution. Il suffit donc d'une pile de booléens pour conserver les valeurs successives de la parité de k, et les restituer à la sortie de la procédure. Le sous-programme s'écrit :

```
F : SI k=0      ALORS u←0 ; v←x
                SINON empiler(impair(k)); k←k/2 ; F ;
                      depiler(t);
                      SI t ALORS u←u+v ; v←2*v
                           SINON v←u+v ; u←2*u
                      IS
    IS
```

Le programme s'écrit : `x←b ; k←n ; F ; resultat←u`

• Transformation en programme itératif:

Il est alors possible d'écrire un programme sans récursivité, sous forme de deux itérations successives :[36]

```
x←b ; k←n ; pile←vide ;
TANT QUE k > 0 FAIRE
      empiler(impair(k)); k←k/2
QT;
u←0 ; v←x ;
TANT QUE nonvide(pile) FAIRE
      depiler(t);
      SI t ALORS u←u+v ; v←2*v
           SINON v←u+v ; u←2*u
```

---

[34] La notion d'«*hypothèse de récurrence*» ou «*invariant de boucle*» est également un aspect important des méthodes développées dans l'ouvrage ; elle sera présentée dans la partie traitant de l'itération : chapitre 5 §1.4.

[35] Considérons une procédure F(k), dont le paramètre k est passé par valeur (ce qui lui donne un statut de paramètre formel). A chaque appel de F(k/2), la variable k reçoit la valeur k/2, et peut être à nouveau modifiée au cours de la procédure. La valeur de k au moment de l'appel doit pouvoir être restituée après l'exécution de la procédure C'est pourquoi, dans le cas de n appels récursifs successifs, les n valeurs sont conservées dans une pile LIFO : la valeur de k "empilée" au moment de l'appel d'une procédure, est "dépilée" après l'exécution L'élimination des paramètres consiste donc à transférer du langage au programmeur, la création et la gestion de cette pile.

[36] L'élimination des appels récursifs permettant la transformation en itération n'est pas triviale, sauf dans le cas d'un appel récursif terminal. Dans le cas de l'exemple, on a un seul appel, central:

F : si <condition> alors A_0 sinon A F B is

En interprétant ce schéma récursif en terme d'exécution, on s'aperçoit que A est répétée, une instance de B étant chaque fois laissée en attente, jusqu'à <condition>, donc ensuite on a autant d'exécutions de B qu'il y a eu d'exécutions de A, ou, ce qui revient, au même, on répète B jusqu'à ce que la pile construite à l'étape précédente pour les paramètres en attente soit vide. D'où les deux itérations successives qui résultent de la transformation. Repeter A jusqu'à <condition> A_0 Repeter B jusqu'à <pile vide>. Le passage de la forme Repeter ... jusqu'à <condition>à la forme TANT QUE <non condition> permet de prendre en compte le cas où <condition> est vérifiée dès l'entrée dans l'itération.

```
        IS
QT;
resultat←u
```

• Représentations de la pile de booléens et transformation du programme:

Jusqu'à cette étape, la pile est envisagée de façon abstraite. La réalisation de la structure "pile de booléens" intervient maintenant; dans une première représentation, on choisit d'utiliser un nombre entier : ce nombre entier est celui dont l'écriture en base 2 est obtenue en empilant par la droite des 1 pour la valeur booléenne vraie, et des 0 pour la valeur booléenne fausse. La valeur 1 de l'entier correspondant à une pile vide, les opérations sont ainsi implémentées :

- `pile←vide  s'exprime  p←1`
- `empiler(t)  s'exprime  p←2*p ;SI t ALORS p←p+1 IS`
- `depiler(t)  s'exprime  t←-impair(p) ; p←p/2`
- `vide(p)  s'exprime  p=1`

Ceci conduit à un programme, où la première boucle `TANT QUE` donne à `p` la valeur de l'entier dont la représentation en base 2 est la représentation symétrique de celle de `n` (le multiplicateur donné), précédée à gauche d'un bit 1. D'où l'idée que la représentation en base 2 de `n` est un élément de résolution : mais au lieu d'empiler à droite, il convient d'empiler à gauche, pour qu'à la fin de la première boucle `TANT QUE`, `p` ait pour valeur `n` (et on pourra donc supprimer ce calcul itératif, et le remplacer par une affectation) ; dans cette représentation, l'entier `p` ne suffit pas à décrire la pile, si l'on ne connait pas le nombre de booléens empilés (l'entier 110 peut représenter l'empilement dans l'ordre de 1,1,0, aussi bien que celui de 0, 1,1,0 ...). La pile est donc constituée de la donnée de l'entier `p` , et de sa hauteur `h` qu'il est commode de donner sous forme d'un entier prenant comme valeurs les puissances de 2 successives :

- `pile←vide  s'exprime  h←1 ; p=0`
- `empiler(t)  s'exprime  SI t ALORS p←p+h IS ; h←h*2`
- `depiler(t)  s'exprime  h←h/2 ; t← (p ≥ h);`
  `SI t ALORS p←p-h IS`
- `vide(p)  s'exprime  h=1`

`p` étant initialisé à `n`, il suffit, dans la première boucle `TANT QUE` de calculer la hauteur `h` ; le programme prend la forme définitive:

```
h←1 ;
TANT QUE h ≤ n FAIRE h←h*2 QT;
u←0 ; v←b ;p←n;
TANT QUE h ≠1 FAIRE
    h←h/2 ;
    SI p ≥ h ALORS u←u+v ; v←2*v ; p←p-h
           SINON v←u+v ; u←2*u
    IS
QT ;
resultat←u
```

4.2.2 Les choix didactiques dans cet exemple

• le gain d'efficacité n'est pas l'essentiel : l'exécution de la fonction récursive de départ comporte dans le pire des cas (c'est à dire le cas où $n = 2^a - 1$), un nombre d'appels de la fonction égal à la somme des a premiers entiers, c'est-à-dire `a*(a-1)/2` ; la complexité au pire est donc de l'ordre de $(\log_2(n))^2$. Mais dès la première transformation, la complexité devient de l'ordre de $\log_2(n)$; la toute dernière transformation évite, dans la première boucle, le calcul de la parité de k, donc apporte un léger gain d'efficacité.

• la méthode vise à donner, pour une large classe de problèmes, des moyens de création des algorithmes: en effet, l'algorithme résultat final n'employant ni la récursivité, ni de type structuré (pile), trouverait sa place dans un exposé élémentaire sur l'itération. Le but de l'ouvrage, tel qu'il apparaît ici n'est pas d'exposer des algorithmes "tout-fait", mais de donner des moyens de les obtenir.

• la méthode implique une relation particulière entre les algorithmes et les structures

de données, impliquant de jouer sur le type abstrait et sur les représentations : la façon dont la pile de booléen apparait puis disparaît est tout à fait intéressante, si on la compare au traitement de l'arbre dans l'exposé de l'algorithme de sélection par tournoi (paragraphe 3.3). En premier lieu, la pile est une organisation des données qui découle ici clairement d'un choix de transformation de l'algorithme (la passage d'un sous-programme récursif à un sous-programme itératif), alors que les raisons de l'apparition de l'arbre dans l'algorithme de sélection par tournoi de [GAUDEL SORIA FROIDEVAUX 86] ne sont pas indiquées. Ensuite, dans le même exemple, les auteurs, en cohérence avec le reste de l'ouvrage n'envisagent pas les possibilités de représentation de l'arbre ; dans l'exemple de [ARSAC 83], au contraire, on voit comment la discussion de la représentation de la pile est un élément essentiel de la transformation.

- la relation entre les algorithmes et les structures de données met en jeu *les significations* des éléments internes à la représentation: en effet la dernière transformation résulte du *choix* d'une représentation parmi deux possibles, en reconnaissant une représentation où à un moment de l'exécution, une variable servant à la représentation coïncide avec une des variables du problème.

Pour résumer et conclure sur ce point, nous dirons que cet exemple illustre la façon dont la construction d'algorithmes doit considérer de façon liée le double aspect abstraction-représentation, et prendre en compte la signification, dans le problème, des éléments internes aux représentations. Ceci s'oppose à une conception de la programmation abstraite où la représentation apparaîtrait comme un aspect secondaire imposé par les nécessités du codage en machine.

4.3 Essai de caractérisation de la démarche pédagogique employée dans l'ouvrage de J. ARSAC

Nous tentons ici de relier les options de J.ARSAC au travail théorique de G. BROUSSEAU ([BROUSSEAU 86] voir note en § 3.3) élaboré pour la didactique des mathématiques. Il ne s'agit évidement pas de dire que J.ARSAC s'inspirerait du travail de G. BROUSSEAU, mais plutôt de voir comment ce travail théorique permet de caractériser la démarche pédagogique exposée par J.ARSAC. Nous pensons en particulier au travail de «*recontextualisation*» auquel doit se livrer le professeur qui tente, à partir d'un savoir constitué, de définir une tâche pour l'élève visant à l'acquisition de ce savoir. G. BROUSSEAU indique que c'est à l'élève de (re)construire ce savoir à partir de situations proposées par le professeur. Il importe donc que le professeur construise des situations où le savoir visé apparaisse comme permettant la résolution de problèmes. Il devra également créer les conditions permettant à l'élève de concevoir le savoir visé en dehors du(des) contexte(s) recréé(s) par le professeur.

J.ARSAC compare la construction de programmes à la résolution de problèmes de géométrie par les méthodes *analytiques* ; ces méthodes consistent à prendre un repère cartésien et à l'aide des équations des objets géométriques, à transformer le problème en résolution d'équations. Donnons deux aspects où ce parallélisme nous paraît éclairant:

- La programmation analytique, comme la géométrie analytique donne un moyen par lequel aborder le problème, en l'absence d'"intuition" d'une solution.
- Elles plongent le problème dans un cadre calculatoire, où la signification des éléments du calcul sert de guide. Dans le traitement analytique d'un problème géométrique, en effet, la solution n'apparaît pas de façon automatique après que les équations aient été posées; pour mettre en place un système de coordonnées adaptées, pour conduire les calculs, une "intelligence" des éléments du calcul est nécessaire: il faut être capable de donner une *signification* aux objets algébriques, c'est-à-dire de mettre en relation le cadre algébrique (les variables, les expressions, les équation) et le cadre géométrique (objets géométriques, transformations..). Le cadre calculatoire de la programmation analytique est constitué de correspondances de haut niveau entre éléments algorithmiques (les transformations d'algorithmes) et de correspondances entre structures de données (les représentations). Ces correspondances font interagir les traitements et les structures de données, ces dernières intervenant sous les deux aspects abstraction et représentation et il est nécessaire, pour

conduire les calculs, de dégager la *signification* des éléments qui y apparaissent

Dans l'ouvrage de J.ARSAC, un algorithme n'est donc pas présenté d'emblée à un niveau achevé d'efficacité et de simplicité ; il est replacé dans le contexte d'un problème à résoudre, d'un procédé de calcul à réaliser en machine ; à travers la transformation d'algorithmes, l'étudiant approche les structures algorithmiques générales et leurs rapports. Le travail de «*recontextualisation*» qui incombe au professeur, consiste à rechercher un procédé de calcul, à partir duquel construire un programme. La recherche d'une expression itérative, la plus simple, la plus élégante, et la plus efficace possible, conduira l'étudiant à dépouiller son texte de programme des traces laissées par les transformations successives qu'il a subi. L'étudiant se livre donc bien à un travail de «*redécontextualisation*». Le travail de recontextualisation s'étend jusqu'aux structures de données ; en effet celles-ci ne sont pas étudiées pour elle-même ; elles apparaissent et se transforment (et même disparaissent) quand la nécessité de l'évolution de l'algorithme l'impose et leur transformation implique de jouer sur les deux aspects : structure abstraite et représentation. La recontextualisation des structures de données s'opère ainsi comme effet de bord de la méthode de construction d'algorithmes.

4.4 Les capacités attendues de l'étudiant dans la programmation analytique

Il est clair que les méthodes de J.ARSAC ne s'adressent pas à des étudiants débutants, et nous retenons de ce paragraphe le fait qu'une démarche de recontextualisation est possible en informatique, et non l'idée de transposer ces méthodes pour des débutants. Un des buts de ce chapitre est d'examiner les spécificités des débutants, et nous avons indiqué qu'une première manière d'aborder cette question est d'examiner les capacités requises chez un étudiant pour aborder un enseignement universitaire, un débutant pouvant se caractériser en premier lieu par l'absence de tout ou partie de ces capacités.

Il nous semble que la programmation analytique consiste à transformer l'algorithme, par adaptations successives à des dispositifs distincts, des schémas de transformation étant mis à la disposition des étudiants. Le dispositif initial est une machine récursive, c'est-à-dire disposant d'un langage fonctionnel récursif. La première étape de dérécursivation conduit à un algorithme implémentable sur une "machine à pile", c'est-à-dire un dispositif itératif disposant d'une pile de booléens. L'étape finale concernerait un dispositif itératif ayant seulement des capacités de traitement sur des entiers. L'étudiant doit d'une part concevoir des traitements et un mode d'organisation des données pour chacun de ces dispositifs, et d'autre-part pouvoir décider d'une organisation des données en fonction des nécessités de l'algorithme et ce sous forme abstraite (indépendamment d'une représentation), puis envisager des représentations en repérant celle où les objets introduits ont une signification dans le problème. La programmation analytique suppose donc que l'étudiant dispose de **Systèmes de Représentation et de Traitement**[37] adaptés à chacun de ces dispositifs, c'est à dire de représentations opératoires de ces dispositifs (et nous avons vu qu'ils sont divers, et éloignés des représentations spontanées issues de l'intuition sensible), et surtout qu'il soit capable en s'aidant des schémas de transformation, d'élaborer des codes de transfert d'un système à l'autre.

5. Conclusions

Nous tentons ici de répondre aux questions posée en début de chapitre (paragraphe 1.3).

5.1 Qu'est ce que la structuration des données?

Du point de vue du dispositif, c'est un ensemble d'éléments du langage, au même titre que les structures algorithmiques ; ces éléments peuvent être vus sous deux

[37] CF [HOC 77] que nous présenterons au chapitre **2** §1.

aspects :

- le "type abstrait" se définit par l'ensemble des valeurs prises par un type de données, et par l'ensemble des définitions algébriques des fonctions et opérations sur ces valeurs,
- les "représentations" permettent que les valeurs d'un type de données A soient formées à l'aide des valeurs d'un type B, que les fonctions du type A soient construites comme des algorithmes utilisant comme primitives les fonctions du type B. Les représentations constituent un lien (abstrait) entre types abstraits, et interviennent dans la programmation effective pour l'implémentation des types de données définis dans la solution, à l'aide des types pré-définis dans le langage utilisé.

Nous avons dit au début de ce chapitre que nous étions à la recherche d'éléments unificateurs permettant de guider des stratégies d'apprentissage. Il apparaît que ce double aspect abstraction-représentation est de nature à apporter des éléments sur les situations d'apprentissage concernant les données informatiques. Nous en donnons trois exemples, sans chercher à être exhaustif:

- un élément unificateur doit permettre d'**éviter des constructions artificielles**: il nous semble par exemple, qu'un enseignement qui serait trop lié aux particularités d'implémentation d'une structure dans un langage donné (par exemple: les chaînes de caractère en LSE, ou en BASIC, ou les listes en **LISP**) cacherait la structure abstraite, et donnerait aux représentations des caractéristiques trop spécifiques. Mettre l'accent sur un type abstrait assez général permet au contraire des acquisitions moins dépendantes du langage de programmation réellement utilisé et évite donc de s'enfermer dans un débat sur les mérites respectifs des langages aux différents niveaux. Nous examinerons les conséquences de ce point de vue au chapitre **5** (§ 2.1) sur l'acquisition des chaînes de caractère, et au chapitre **11** §2 sur l'acquisition des booléens.
- Le double aspect abstraction-représentation permet d'apprécier le **niveau conceptuel** de la notion de structure de données. Nous voulons dire par là que toutes les notions abordées dans un enseignement ne sont pas à mettre sur le même plan, par exemple en géométrie, le tracé d'une figure à partir d'une liste de consignes d'une part la démonstration d'un théorème d'autre part ne sont pas du même niveau conceptuel, et que le fait de replacer une notion dans le savoir constitué grâce à ce que nous avons appelé un élément unificateur permet d'apprécier son niveau; pour reprendre l'exemple de la soustraction (voir paragraphe 1.3), le fait de la relier au concept général d'équation montre qu'elle est d'un niveau conceptuel plus élevé que le simple apprentissage d'un mécanisme. La définition d'un type abstrait pour une donnée d'un problème, revient à la définition d'un système formel, avec un alphabet de symboles, des régles de formation des expressions et un système d'axiomes. En tant que tâche pour l'élève, la définition des types pour un problème, et son utilisation pour un programme suppose que l'élève construise des hypothèses, et utilise des objets selon les hypothèses, et non selon leurs propriétés intuitives ; elles s'apparentent donc à l'élaboration d'une démonstration mathématique, et sont d'un niveau conceptuel comparable. La représentation est la mise en relation de deux systèmes formels ; du point de vue de son niveau conceptuel, on peut donc la mettre en relation avec la "construction" d'une structure dans une autre en mathématiques, par exemple la construction des nombres complexes comme couples de réels.
- Le double aspect abstraction-représentation peut permettre de **caractériser la façon dont les structures sont abordées** en situation pédagogique. Les structures numériques, par exemple, sont utilisées de façon abstraite lors de l'initiation, en s'appuyant sur la connaissance de ces structures présente en principe chez les élèves ; c'est seulement à un niveau plus avancé qu'on examinera avec les étudiants les questions d'implémentation pour une machine disposant de registres de taille limitée. La structure abstraite des chaînes de caractères est par contre d'un genre nouveau pour les élèves débutants en ce sens qu'elle n'est ni numérique, ni géométrique : les élèves ne peuvent donc pas les envisager de façon abstraite; on s'appuie donc sur des représentations faisant appel à des dispositifs généralement peu explicites.(nous approfondirons ce point au chapitre 3). Quand on travaille sur des structures dynamiques, avec des étudiants plus avancés, il n'est pas rare qu'on utilise, pour guider la réflexion, des représentations pour une machine plus ou moins

virtuelle possédant une mémoire dont les cellules sont accessibles par des adresses (nous avons vu au § 2 que D.KNUTH décrit de façon précise la machine sur laquelle il s'appuie pour les représentations). Cette utilisation de la représentation, comme lien entre structures a son analogue en mathématiques, où les structures en voie d'acquisition peuvent être considérées soit pour leurs propriétés, soit à l'aide de représentations dans des structures déjà connues des élèves : ainsi, les nombres complexes peuvent être considérés pour leurs propriétés, ou en représentation cartésienne (a + i b, a et b réels) ou polaire ($\rho e^{i\theta}$). On sait que la représentation cartésienne peut aider les élèves dans les acquisitions, mais qu'ils doivent apprendre à travailler sans, ou à choisir pour chaque problème la représentation adaptée : par exemple, il serait maladroit de résoudre $z^2=-1$ en passant à la forme cartésienne. Le double aspect abstraction-représentation génére donc une variable didactique.

5.2 La structuration des données, du point de vue du sujet ; le cas du débutant en programmation

Considérer un type de façon abstraite suppose chez le sujet la capacité d'isoler les éléments pertinents d'une situation et leurs propriétés. Nous avons souligné que la définition complète d'un type abstrait pour une donnée d'un problème, et son utilisation dans un programme s'apparentent à une démonstration mathématique. On se trouve donc dans une situation analogue à celle de l'enseignement de la géométrie, où les élèves ne peuvent prendre conscience que très progressivement de la nécessité de raisonner sur des éléments abstraits (c'est-à-dire considérés seulement pour leurs propriétés postulées) et non sur des objets intuitifs. L'activité de représentation suppose quant à elle l'existence chez le sujet de plusieurs **Systèmes de Représentation**[38] **de Traitement (S.R.T.)**, la capacité de les mettre en relation et d'élaborer des procédures de passage de l'un à l'autre.

C'est pourquoi, des sujets débutants, faute d'avoir été placés dans des situations impliquant de réfléchir sur une structure abstraite, et impliquant le choix d'une représentation, peuvent ne pas différencier les deux aspects dans la conception qu'ils ont des objets, ce qui correspond à un niveau d'intériorisation du langage où les **Systèmes de Représentation et de Traitement (S.R.T.)** sont peu différenciés et peu hiérarchisés. Par exemple, la conception des nombres entiers pour un débutant pourra correspondre aux entiers naturels des mathématiques, ce qui est correct d'un point de vue abstrait, mais ne prend pas en compte la nécessaire limitation de l'ensemble des valeurs dans un dispositif informatique, ni la nécessité du choix d'une représentation dans un langage de programmation, choix qui conditionne l'efficacité du traitement algorithmique : tant qu'il s'agit d'implémenter un calcul simple sur des entiers, cette question pourra rester dans l'ombre, mais s'il s'agit par exemple de traiter le numéro INSEE à 13 chiffres (voir chapitre 9), ou de trouver un contre-exemple à la conjecture de FERMAT dans un langage codant les entiers sur deux octets, alors les questions de représentation vont se poser et solliciter une différentiation entre le **S.R.T.** habituel des nombres entiers, et un **S.R.T.** adapté à l'implémentation d'algorithmes, ainsi que la recherche de procédures de passages entre ces **S.R.T.**

Nous sommes donc conduit à penser que, si le double aspect abstraction-représentation est bien l'élément unificateur recherché, le premier obstacle que rencontrera un débutant est la **nécessité de se dégager d'une conception intuitive des données pour envisager ce double aspect.** La conception intuitive pourra être très dépendante de la nature de l'objet. Pour les types numériques, compte-tenu des acquisitions antérieures d'au moins une partie des élèves, c'est l'aspect abstrait qui sera le plus facile à envisager, les contraintes d'implémentation sur un dispositif pouvant ne pas être perçues. Nous étudierons, en parties II, III et IV, à travers l'observation d'élèves en situation de résolution, leurs

[38] Il s'agit ici des représentations mentales résultant, chez le sujet, de l'intériorisation des propriétés des objets présents dans un domaine de tâches et de leurs lien avec des objets d'autres domaines.

conceptions intuitives concernant les chaînes de caractères et les booléens pour lesquels ils ne disposent pas d'acquis antérieurs.

D'autre-part, dans l'analyse, telle que nous l'avons menée dans ce chapitre, les structures de données sont apparues internes au programme, sans que se pose la question de l'interaction entre le programme et un utilisateur. Montrons qu'une difficulté se pose et qu'elle peut ne pas avoir été aperçue par des auteurs, y compris dans des ouvrages explicitement destinés à des débutants. «*Premières leçons de programmation*» [ARSAC 80][39] est un de ces ouvrages. Il s'organise comme une série de problèmes commentés marquant une progression dans l'acquisition des structures algorithmiques. Le premier problème, qui s'adresse en principe à des débutants complets est le calcul, un millésime étant entré, du jour de la semaine où "tombe" la fête de Noël l'année de ce millésime. Le résultat pose un problème de représentation, car si le traitement numérique qui s'impose dans le problème conduit à un résultat numérique (par exemple un nombre de 0 à 6), l'idée que se fait une personne non avertie de la communication entre le programme et l'utilisateur est que ce résultat est un libellé explicite ("Lundi", "Mardi",...). Pourtant, la question du type du résultat est évacuée par l'auteur dès le second paragraphe par une recommandation tout à fait dans l'esprit de la programmation abstraite : «*On laissera cette question en attente le plus longtemps possible...*». Cette recommandation suppose de façon évidente que le lecteur connaît l'existence de procédures de passage d'une représentation numérique des jours de la semaine à une représentation sous forme de chaîne, ce qui n'est certainement pas le cas d'un débutant ; deux procédures de ce type sont indiquées, mais bien plus loin, après la discussion du traitement algorithmique, et sont exposées plus comme des détails de présentation que comme une question importante de programmation.

Il est certain que cette difficulté n'en est pas une pour un programmeur confirmé qui connaît des procédures permettant de compenser l'impossibilité, ou le caractère peu commode des entrées-sorties pour certains types de données : ainsi, les booléens en PASCAL, ne peuvent être l'objet d'une entrée directe au clavier, mais un programmeur confirmé sait qu'un booléen peut recevoir une valeur suite à la lecture d'une chaîne, et à une affectation ; par exemple :

```
Var rep: chaine;
    Masculin:boolean;

begin
   writeln('Etes-vous de sexe masculin (M) ou féminin (F)');
   readln(rep);
  Masculin:=(rep='M')
end
```

Par contre, les questions d'entrée-sortie (lecture des données, affichage des résultats) et plus généralement d'interaction entre le programme et l'utilisateur pourront être, pour le débutant, de véritables problèmes, et donc être, pour lui, constitutives d'une structure de données. Dans un enseignement destiné à des débutants, les questions relatives à l'interaction entre le programme et l'utilisateur, qui font intervenir des questions de représentation, ne peuvent donc être traitées comme des détails, mais sont bien partie intégrante de la structure.

5.3 L'évolution historique, et les conditions d'apparition des structures de données

Nous avons appris, avec [KNUTH 67] (paragraphe 2.2), que l'apparition d'un type de structuration des données peut se dater historiquement, et qu'elle se produit généralement comme solution à une classe de problèmes, avant que la structure ainsi créée soit considérée comme un objet informatique et étudiée comme tel. On pourrait dans cet esprit, rechercher les problèmes dans lesquels un type de structuration de données apparaît comme une technique, et les conditions didactiques dans lesquelles les élèves pourrait ensuite considérer cette structure de façon plus générale : il nous semble que les files, les tableaux par exemple, pourraient être construits par les

[39] J.ARSAC ("Premières leçons de programmation" , CEDIC 1980)

élèves en réponse à des problèmes à l'aide des types connus, avant que la structure soit considérée et étudiée de façon générale.

5.4 Les choix didactiques liés aux rapports entre structures de données et structures algorithmiques

Les conceptions présentées dans [GAUDEL SORIA FROIDEVAUX 86] (paragraphe 3.3) conduisent de façon schématique, à privilégier les définitions abstraites des types de données dans l'activité de programmation, la représentation étant considérée comme une tâche annexe. Au contraire, [ARSAC 83] fait jouer les deux aspects (abstraction-représentation) des structures de données au cours des transformations qu'il opère sur les algorithmes (paragraphe 4.3). Nous retiendrons donc l'intérêt du raisonnement sur des données abstraites, mais aussi que l'activité de représentation ne doit pas être éliminée si l'on veut pouvoir faire évoluer de façon conjointe algorithme et structuration des données.

Les deux ouvrages dont nous avons développé l'étude, font apparaître aussi que ces conceptions sont reliées à des stratégies pédagogiques : stratégie d'exposition qui conduit à présenter les notions de façon décontextualisée dans le cas de [GAUDEL SORIA FROIDEVAUX 86], stratégie de construction qui vise à donner des outils à l'étudiant dans le cas de [ARSAC 83]. L'étude pourrait être prolongée par la lecture d'autres ouvrages, en particulier, [PAIR MOHR SCHOTT 88] qui propose une démarche où la structuration des données, et la construction des structures algorithmiques progressent de pair, l'une s'appuyant sur l'autre.

Nous avons indiqué au début de ce chapitre (paragraphe 1.2 question 4) que ce repérage de choix didactiques avait pour nous un intérêt s'il révèle des choix pertinents pour des cursus variés, comprenant en particulier l'initiation. Montrons par un exemple que les rapports entre les apprentissages concernant la structuration des données d'une part, et les structures algorithmiques d'autre part, sont susceptibles de conduire à des choix pertinents aussi au niveau de l'initiation: en classe de Seconde option informatique, le choix est fait par de nombreux professeurs, de faire travailler les élèves d'abord sur les structures algorithmiques (séquentialité, alternatives, itération...) en considérant seulement le type numérique, puis de présenter par exemple les chaînes de caractères, et de faire fonctionner sur les chaînes les structures algorithmiques présentées antérieurement: beaucoup de ces professeurs font l'hypothèse que les algorithmes s'acquièrent indépendamment des objets sur lesquels ils portent, et que les acquisitions se transfèrent facilement d'un type d'objet à l'autre. On peut envisager au contraire de faire travailler les élèves dès le départ sur des objets de type chaîne, aussi bien que sur des nombres, avec une structure algorithmique très élémentaire, puis de progresser dans l'acquisition des structures algorithmiques à la fois dans les types numériques et dans le type chaîne. Dans ce cas, la variété des types disponibles donne des possibilités nouvelles; si, en suivant [PAIR MOHR SCHOTT 88], on se propose de déduire la structure algorithmique du type du résultat, on pourra considérer des problèmes où le résultat est un ordinal (alors l'algorithme pourra être une itération sur des ordinaux successifs), et des problèmes où le résultat est une chaîne (alors l'algorithme pourra être la construction par concaténation itérée de cette chaîne).

Dans la résolution d'un problème, comme dans la présentation d'un algorithme, il apparaît donc important de lier le choix et la construction d'une structuration des données à la construction de la structure algorithmique, l'idée d'une structuration "naturelle" des données ayant plutôt comme résultat de masquer ce lien. Nous retenons de [ARSAC 83] le fait qu'une démarche de *recontextualisation* peut passer par la résolution de problèmes où l'algorithme et les structures de données évoluent de façon coordonnée en jouant sur les deux aspects : structure abstraite et représentation. La recherche d'une recontextualisation de ce type apparaît une direction d'investigation prometteuse pour les différents concepts et les différents niveaux de l'enseignement de l'informatique.

Chapitre 2 Le sujet et l'activité de programmation

représentations mentales
fonctionnement mental
signification des concepts informatiques

Une thèse en didactique utilise nécessairement des outils théoriques, des modes d'approche des problèmes, des résultats élaborés par d'autres chercheurs. La didactique de l'informatique est moins développée que la didactique d'autres disciplines; elle peut cependant s'appuyer sur des travaux dans deux directions: d'une part des **recherches en psychologie du travail** centrées sur l'activité de programmation auxquelles a conduit le développement de l'informatique dans le domaine professionnel en France et à l'étranger; d'autre part, quelques **recherches en didactique** ont été menées en France, à l'occasion de l'introduction d'un enseignement de l'informatique en lycée (nous présenterons cet enseignement au chapitre 4), et dans d'autres pays à l'occasion de la construction de curriculum expérimentaux. Nous étudions par conséquent dans ce chapitre deux comptes rendus de recherche en psychologie du travail , et deux comptes rendus de recherches dirigées vers l'enseignement, notre but étant de préciser les outils théoriques, les problématiques, les résultats que nous utiliserons dans la suite. Nous avons choisi ces textes pour le caractère général des questions qu'ils posent, et pour la variété des outils et problématiques qu'ils présentent.

Nous intéressant à une problématique relativement nouvelle, nous n'avons pas cherché à faire une présentation exhaustive de travaux dans ces deux domaines, pensant qu'un choix limité permettrait de mettre en évidence les outils nécessaires tout en permettant une étude en profondeur. Dans cet esprit nous n'avons pas retenu des recherches menées sur de jeunes enfants (par exemple [MENDELSOHN 87][40]) qui nous ont semblé porter autant sur l'acquisition de capacités logiques générales, que sur l'appropriation de concepts informatiques. Au contraire les recherches en psychologie du travail que nous étudions ci-dessous, se centrent sur les apprentissages en informatique, et même de façon plus restrictive, sur l'apprentissage de la programmation, et nous ont donc parues plus utilisables pour notre problématique.

Dans les chapitres suivant, sur des points plus particuliers, nous ferons référence à d'autres recherches en didactique: en introduction, nous avons présenté des résultats de [DAGDILELIS BALACHEFF CAPPONI 91], au chapitre **3**, à propos de LOGO et du type de problème sur lequel il permet de construire les apprentissages, nous ferons référence à [SAMURCAY ROUCHIER 90][41] et [HILLEL 86][42] , et au chapitre **5** (partie Annexes) nous présenterons des études en didactiques concernant l'acquisition de l'itération (LABORDE, BALACHEFF, MEIJAS 1985] et [ROGALSKI 84])

- Nous étudions en première partie «*Role of mental representation in learning a programming language*» [HOC 77], qui précise la notion de représentation mentale du dispositif informatique, et met en évidence des degrés dans l'apprentissage du langage du dispositif.
- La seconde partie présente l'article: «*Conditional program statements and their comprehensibility to professionnal programmers*» [GREEN 77]. T.R.G. GREEN s'est intéressé dans la première partie des années 70 à la lisibilité d'assertions

[40] MENDELSOHN P. (1987) «Apprentissage d'un système de commande informatisé et organisation des connaissances chez l'enfant» *Psychologie Française* n°4

[41] SAMURCAY R. ROUCHIER A. (1990) «Apprentissage de l'écriture et de l'interprétation des procédures récursives» *Recherches en didactique des Mathématiques*, Vol 10 2.3. Editions la Pensée Sauvage

[42] HILLEL J. (1986) «Mathematical and Programming concept in a restricted LOGO environnement» *Recherches en didactique des Mathématiques*, Vol 6/2.3 Editions la Pensée Sauvage

conditionnelles par des programmeurs professionnels, de façon évaluer le meilleur style d'assertion et contribuer ainsi à l'amélioration des langages de programmation. Nous relevons dans ce texte des éléments intéressants concernant le fonctionnement cognitif d'un sujet interprétant le texte d'un programme.

- L'article étudié en troisième partie s'intitule «*The Benefit of Microwords in learning computer programming*» [COHORS-FRESENBORG 87] et se rapporte à l'enseignement de la programmation auprès d'élèves d'age scolaire. Il s'agit d'un texte rapportant une analyse des processus d'acquisition chez les débutants en informatique, et un travail d'ingénierie didactique. Même si ce texte est assez éloigné des objectifs de cette thèse, en ce qui concerne les concepts dont l'acquisition est recherchée, il nous a semblé intéressant de l'étudier car il représente une direction de recherche et d'expérimentation assez divergente de ce qui se pratique dans l'enseignement français : les représentations mentales du fonctionnement du dispositif en tant que réalité matérielle sont en effet privilégiées dans cette direction de recherche.
- L'article étudié en quatrième partie, «*Signification et fonctionnement du concept de variable informatique chez des élèves débutants*» [SAMURCAY 85], est un compte-rendu de recherche en didactique dont la problématique est proche de celle de cette thèse : en effet, elle porte sur les processus d'acquisition des concepts, avec un terrain expérimental proche de celui que nous adopterons pour cette thèse. A la suite de l'étude de cet article, nous nous interrogeons en particulier sur l'effet d'un changement dans le type des données sur les conditions d'acquisition des concepts.
- La cinquième partie tire les conclusions du chapitre.

1. Les représentations mentales dans l'activité de programmation (J.M. HOC)

J.M. HOC introduit dans cet article un cadre théorique pour l'étude du travail d'analystes programmeurs en situation professionnelle ; ce cadre a, à notre connaissance, peu évolué depuis la parution de l'article, et s'est révélé pertinent pour un ensemble très large de situations où un sujet est confronté à une tâche de programmation d'un dispositif automatique, parmi lesquelles les situations d'apprentissage où l'on vise à faire acquérir à un programmeur débutant un (des) concept(s) informatique(s) par la résolution de problèmes de programmation[43]. Le texte de J.M. HOC précise les éléments théoriques liés aux représentations mentales, puis décrit une expérimentation auprès d'analystes programmeurs en situation de résolution d'un problème de programmation. Nous rappelons dans un premier paragraphe ceux de ces éléments théoriques qui sont nécessaires dans notre thèse ; dans un second paragraphe, nous rappelons les résultats de l'expérimentation, puis nous donnons des conclusions en rapport avec les objectifs de notre thèse.

1.1. Eléments théoriques

L'activité de programmation suppose un sujet (le programmeur) et un dispositif (la machine pour laquelle il programme). Sous certaines conditions, une tâche de programmation peut-être un problème pour le sujet.

- **Le dispositif** est une machine capable d'opérer certaines actions sur l'environnement selon des régles définies ; ces actions et régles doivent pouvoir être décrites objectivement et constituent «*le langage du dispositif*». Ces langages sont organisés hiérarchiquement selon la relation d'abstraction: ainsi, un langage d'assemblage virtuel comme l'assembleur `MIX` construit par [KNUTH 67] (présenté au chapitre 1) pour coder les algorithmes et représenter les données, décrit un dispositif général permettant de rendre compte du fonctionnement d'une classe de machines réelles, ayant chacune son propre langage d'assemblage, mais disposant de

[43] J.M. HOC a présenté, dans son ouvrage «*Psychologie cognitive de la planification*» Editions PUG ([HOC 87]) une analyse de l'activité du sujet dans la résolution de problèmes impliquant des domaines cognitifs variés. L'étude de la **planification** lors de la résolution de problème menée dans cet ouvrage intéresse directement la didactique de l'informatique. Nous prenons en compte un de ses aspects au §1.3.

principes communs.

- **Les Systèmes de Représentation et de Traitement** sont des constructions mentales élaborées par le sujet : confronté à un dispositif donné, il doit progressivement apprendre (intérioriser) le «*langage*» de ce dispositif et on dit qu'il s'est construit une représentation du dispositif (ou plus précisément un «*Système de Représentation et de Traitement*», en abrégé dans la suite, un **S.R.T.**) s'il est capable de faire fonctionner le dispositif mentalement selon les régles, et de prévoir le résultat d'une série d'actions du dispositif. Ce **S.R.T.** explique les conduites du sujet face au dispositif. Le sujet rencontre toute une série de dispositifs (du plus proche, comme son propre corps, au plus abstrait comme par exemple l'Algèbre en tant que système de régles de calcul formel). Pour chacun de ces dispositifs, il construit un **S.R.T.** Comme les langages des dispositifs, les **S.R.T.** sont organisés hiérarchiquement, des moins abstraits aux plus abstraits : un exemple d'une telle relation hiérarchique est la réponse à donner à un incident sur une machine dans l'industrie : le **S.R.T.** du sujet peut lui permettre seulement d'intervenir directement sur la machine ; un **S.R.T.** plus évolué lui permettra d'agir par l'intermédiaire d'une télé-commande ; à un niveau supérieur il pourra indiquer par communication orale ou gestuelle à un autre opérateur comment intervenir. L'activité de représentation consiste pour le sujet dans la création de codes lui permettant le passage entre des **S.R.T.** de niveau d'abstraction différents.
- **Une tâche de programmation** consiste, pour un dispositif donné, à construire une représentation de l'état initial et de l'état final compatibles avec le langage du dispositif, et une procédure dans ce langage. Elle est un **problème** pour le sujet s'il est capable de se représenter un état initial et un état final, et une procédure conduisant du premier au second dans un (des) **S.R.T.** dont il dispose, et si cette procédure n'est pas directement traduisible dans le langage du dispositif. Le sujet doit donc, pour résoudre, se livrer à une activité de représentation.

1.2. Les degrés d'intériorisation du langage du dispositif

A la suite d'une étude expérimentale portant sur la résolution par un groupe d'analystes programmeurs d'un problème de programmation, J.M. HOC distingue trois degrés, du programmeur débutant au programmeur confirmé :

1 Au premier degré le sujet dispose d'un **S.R.T.** minimal dans lequel il serait possible de se représenter des algorithmes. Mais dans ce **S.R.T.**, le sujet envisage le processus comme une succession de cas, plutôt que comme un algorithme ; en particulier, les boucles ainsi que l'acquisition et la transmission des données restent implicites. Cette conduite résulte de ce que le sujet n'a pas l'expérience de dispositifs qui nécessitent la donnée d'une procédure complète avant le début de l'exécution, et qui disposent de capacités de contrôle. Le sujet tente généralement une traduction mot-à-mot dans le langage du dispositif ; dans cette tentative, le sujet utilise abondamment la structure alternative, mais faute de structures algorithmiques la tentative n'aboutit pas.

2 Au second degré, le sujet élabore une représentation du problème par une série de sous-processus organisés entre eux : le sujet envisage donc la globalité du problème, et non plus une succession de cas particuliers. Mais le dispositif sous-jacent, capable pour le sujet d'exécuter chaque sous-processus, n'est pas encore un ordinateur. Ceci impose au sujet de procéder à des ajustements locaux, (ajout ou déplacement d'instructions) au moment de la traduction d'un sous-processus dans le langage de l'ordinateur ; ces ajustements locaux, remettent en cause la conception d'ensemble du programme. A ce niveau, le sujet est donc capable de considérer des algorithmes, et non une succession de traitements particuliers. Mais il n'a pas intériorisé suffisamment le langage du dispositif : les actions élémentaires sont conçues dans différents **S.R.T.**, à partir desquels le sujet doit procéder à une traduction dans le langage du dispositif.

3 Au troisième degré, le sujet a correctement intériorisé le langage du dispositif ; il dispose d'un **S.R.T.** construit sur plusieurs niveaux permettant une correspondance directe avec le langage de programmation. Ce **S.R.T.** inclue les actions élémentaires, et la syntaxe du langage, ainsi que les algorithmes couramment utilisés. Il permet au sujet d'obtenir une vue d'ensemble de la solution organisée en

sous-processus, directement compatibles avec le dispositif. Cette vue d'ensemble permet une meilleure représentation des données : en effet dans un premier temps, seules les parties pour lesquelles il n'y a pas de solution évidente sont détaillées ; ainsi, ce sont ces parties les plus difficiles qui commandent le choix d'une représentation des données.

1.3. Les S.R.T., les plans: un modèle du fonctionnement mental et ses conséquences

Bien que le travail de J.M. HOC porte sur des analystes programmeurs en situation professionnelle, les éléments théoriques qu'il apporte, et les résultats de l'étude expérimentale sont tout à fait utilisables pour notre problématique, qui se rapporte à des débutants pratiquant l'activité de programmation en vue de l'acquisition des premiers concepts.

En premier lieu, l'article de J.M. HOC nous permet de "baliser" le processus d'apprentissage dans la durée: tant que le degré 3 n'est pas atteint (c'est-à-dire tant que le sujet ne dispose pas d'un **S.R.T.** construit sur plusieurs niveaux permettant une correspondance directe avec le langage de programmation), il est illusoire de penser obtenir directement du sujet une représentation de la solution en sous-processus ; les sous-processus que peut élaborer un sujet antérieurement à ce degré le seront dans des **S.R.T.** incompatibles avec l'ordinateur, et demanderont des ajustements remettant en cause la conception générale du programme. Des méthodes de programmation s'adressant explicitement à des débutants, par exemple, la «*programmation descendante*» [ARSAC 80][44], ou la «*méthode déductive MEDEE*» [DUCRIN 84][45]) supposent toutes d'envisager a priori un plan d'ensemble du programme rendant possible sans retouches majeures la programmation de chacune de ses parties pour le dispositif informatique ; elles sont donc inapplicables avant ce troisième degré de maîtrise du langage du dispositif. De même, c'est seulement avec le troisième degré que le sujet parvient à une codification efficace des données en jeu dans le problème : à ce degré seulement, il peut traiter d'abord les données en jeu de façon abstraite (pour leurs propriétés, indépendamment d'une représentation), puis arrêter la représentation des données au fur et à mesure de la programmation des sous-processus.

La programmation descendante et la programmation sur des données abstraites ne peuvent donc être mises en oeuvre par le sujet que lorsqu'il dispose d'un ensemble de **S.R.T.** mis en relation et hiérarchisés, c'est-à-dire s'il est capable de raisonner sur un dispositif suffisamment général, et donc au niveau des concepts de base de l'informatique. Par contre ces méthodes n'ont pas de signification pour un débutant raisonnant dans un **S.R.T.** issu directement de son expérience sensible, et lié à un domaine de tâches particulier. La **constitution, à partir de problèmes de programmation, d'un ensemble de S.R.T. s'organisant et se différenciant progressivement** nous semble donc l'objectif prioritaire d'un enseignement à des débutants.

Nous retenons également la notion de Système de Représentation et de Traitement (**S.R.T.**) comme permettant l'interprétation de conduites d'un sujet confronté à une tâche donnée en programmation. Un **S.R.T.** est une connaissance locale du sujet explicitée et mise en forme pour les besoins de l'analyse psychologique ou didactique. Les **S.R.T.** se rapprochent donc des "conceptions" telles que définies par [ARTIGUES 91] [46]. Mais, les **S.R.T.** sont des conceptions particulières, en ce sens

---

[44] «*2.13. LA PROGRAMMATION DESCENDANTE (...) L'idée de la programmation descendante est de construire le programme en commençant par un plan qui ne comporte que l'intitulé de ses principales phases*» Jacques ARSAC ajoute «*Avant de construire une maison, on en fait le plan*». Il nous semble, pour notre part, que le programmeur débutant est souvent dans la situation où on lui demande de construire le plan d'une maison alors qu'il ignore tout de ses habitants, de la façon dont on construit une maison, des contraintes matérielles d'une construction...

[45] DUCRIN A. (1984) *Programmation 1* Dunod. Le chapitre 2 s'intitule "*Discours de la méthode*" et décrit la méthode MEDEE comme une méthode descendante partant du résultat final.

[46] ARTIGUES M. (1991) «Epistémologie et Didactique» *Recherche en didactique des*

qu'ils intègrent des éléments procéduraux et sont donc de véritables "machines mentales", que le sujet fait fonctionner pour ses anticipations et ses vérifications. Un des buts de cette thèse est d'examiner comment les premiers apprentissages en informatique peuvent s'analyser comme processus de construction de **S.R.T.** adaptés à des domaines de tâche de programmation. En particulier, de façon à prendre en compte l'interaction des données et des algorithmes dans le fonctionnement mental, nous considérerons qu'un **S.R.T.** mis en oeuvre par le sujet dans la résolution d'un problème de programmation, possède en tant que "machine mentale" une organisation analogue à celle des dispositifs programmables matériels ou virtuels, où l'on peut distinguer d'une part des structures relatives aux informations (ou objets) manipulés par le dispositif, et d'autre part des structures relatives aux traitements (ou algorithmes) qu'effectue le dispositif sur ces objets. Nous postulerons donc l'existence, au sein d'un **S.R.T.** donné de plan(s) relatif(s) aux données (ou objets) d'une part, et de plan(s) relatif(s) aux traitements (ou actions complexes) d'autre part, et nous nous proposons d'étudier les interactions entre ces plans. Nous précisons ci-dessous le sens que nous donnons à la notion de *plan*.

Selon [HOC 87] un plan est «*une représentation schématique et (ou) hiérarchisée susceptible de de guider l'activité du sujet*»; dans la suite de cette thèse, nous utiliserons cette notion seulement dans un sens restreint: les problèmes dont nous prévoyons la résolution par des débutants étant de peu d'ampleur, nous n'envisageons pas la hiérarchisation des plans, ni par conséquent la planification. Nos *plans* sont donc des guides schématiques que le sujet utilise pour construire son programme, se rapprochant du plan permettant de se repérer dans un bâtiment, plus que de l'objet d'une opération de planification. Ils sont conformes à la définition de [SAMURCAY 87]: «*un ensemble organisé de connaissances qui représentent sous forme générique, des concepts, des procédures, des événements ou des séquences d'événements*».

Le même auteur précise: «*Au moment de leur utilisation, les variables du schéma* (ou plan) *sont particularisées en fonction du contexte évoqué par la situation*». Devant un problème de programmation évoquant pour lui une situation générale, le programmeur fera appel à un plan dont il dispose. Etant suffisamment schématique pour s'appliquer à une classe de situations, ce plan comporte des éléments génériques. Le programmeur devra ensuite tenter de particulariser ces éléments. Comme le remarque [HOC 87], cette généricité des plans permet leur importation d'un S.R.T. à l'autre: «*Une situation donnée peut évoquer directement un SRT relativement pauvre en plans (...) il arrive que le sujet parvienne à évoquer un plan appartenant à un autre SRT, dans lequel il dispose d'un répertoire de plans plus riche*»

En particulier, face à un problème de programmation modélisant une situation connue, le débutant ne dispose pas dans le **S.R.T.** qu'il met en oeuvre pour l'écriture de son programme, de plans directement applicables; il est donc contraint d'importer des plans à partir du **S.R.T.** servant dans la situation connue. Nous verrons que souvent, dans un **S.R.T.** correspondant à une situation familière, les traitements sont liés aux objets, et que donc, en adaptant des *plans* issus de ces **S.R.T.**, le sujet intègre les traitements dans le **S.R.T.** correspondant au dispositif de façon dépendante des objets sur lesquels ils portent. Un des objectifs de cette thèse sera donc de mieux comprendre comment s'effectue cette intégration des traitements, d'examiner quelle confrontation au dispositif informatique conduit à remettre en cause des conceptions des traitements trop liées aux objets, et donc à leur donner une portée plus générale et plus opératoire.

2. Une étude sur la compréhension des écritures conditionnelles (T.R.G. GREEN)

T.R.G. GREEN s'est intéressé dans la première partie des années 70 à la lisibilité d'assertions conditionnelles par des programmeurs professionnels, de façon évaluer le

meilleur style d'assertion et contribuer ainsi à l'amélioration des langages de programmation. Il s'agit donc d'une recherche dont les conséquences ne sont pas a priori du domaine de la didactique. Cependant cette étude apporte des éclaircissements utiles pour notre recherche sur le fonctionnement cognitif d'un sujet confronté à une tâche d'analyse d'un texte de programme. Le fonctionnement cognitif du sujet, c'est-à-dire les schèmes qu'il met en oeuvre pour anticiper les réponses du dispositif constituent un des aspects (lié aux traitements) de la notion de **S.R.T.** rencontrée ci-dessus.

2.1 Les styles d'écritures conditionnelles.

L'auteur définit deux styles :

- Le «*Jumping style*» c'est-à-dire le style utilisant la rupture de séquentialité permise dans certains langages par des instructions de saut (style *GOTO*).
- Le «*Nesting style*» c'est-à-dire le style avec *alternatives emboîtées.*

Considérons avec l'auteur l'écriture d'une conditionnelle à trois cas portant sur la cuisson d'un légume :

• dans le "style *GOTO*"

```
IF "à feuille" THEN GOTO L1
frire
GOTO L3
L1 IF "vert" THEN GOTO L2
bouillir
GOTO L3
L2 griller
L3 finir
```

• dans le style avec *alternatives emboîtées*

```
IF "à feuille"  THEN
                  IF "vert" THEN
                            griller
                           ELSE
                            bouillir
                ELSE frire
```

Pour des conditionnelles où les actions impliquées sont constituées de plusieurs sous-actions,(une sous-action pouvant elle-même être conditionnelle) l'auteur souligne que le style avec *alternatives emboîtées*, implique en général l'utilisation d'un parenthésage de blocs (comme par exemple le `BEGIN...END` de PASCAL, les parenthèses de LISP, ou les crochets de LOGO...). Ci-contre, l'exemple de l'auteur :

```
IF "juteux"    THEN
               BEGIN
                  hacher frire
               END
            ELSE  BEGIN
                    IF "dur"   THEN
                             peler
                            ELSE
                             BEGIN
                              hacher
                              frire
                            END
                  END
```

On notera la nécessité du parenthésage de blocs, en constatant que l'écriture ci-contre est susceptible de plusieurs interprétations, suivant le bloc où l'on situe la seconde instruction "`frire`"

```
IF "juteux" THEN
      hacher frire
            ELSE
      IF "dur" THEN
                 peler
               ELSE hacher frire
```

On peut en effet avoir l'interprétation ci-dessus, mais aussi parmi d'autres, l'interprétation ci-contre:

```
IF "juteux"  THEN
               BEGIN
               hacher frire
               END
             ELSE
             BEGIN
               IF "dur"   THEN
                           peler
                          ELSE
                           hacher
               frire
             END
```

T.R.G. GREEN propose une autre solution à cette question : répéter la condition, mais sous forme négative, et utiliser une instruction de fin de conditionnelle pour marquer la fin de la suite de sous-actions constituant chacune des actions impliquées dans la conditionnelle (*alternatives emboîtées If-Not-End*) ; l'exemple ci-dessus s'écrirait comme ci-contre :

```
IF "juteux" :
     hacher frire
NOT "juteux" :
     IF "dur":
          peler
     NOT "dur" :
          hacher frire
     END "dur"
END "juteux"
```

2.2 L'étude expérimentale

Conduite sur 12 programmeurs expérimentés, elle a consisté en deux épreuves de lisibilité, basées sur la rapidité avec laquelle ces programmeurs produisent une réponse exacte :

- une épreuve consistant, un programme conditionnel et des valeurs de vérité des entrées étant donnés à produire une prédiction du résultat,
- une épreuve consistant, un programme conditionnel et un résultat étant donnés, à produire les valeurs de vérité des entrées nécessaires pour ce résultat.

Les résultats montrent dans les deux épreuves une moins grande lisibilité du "style *GOTO*" ; le style "*alternatives emboîtées* avec parenthésage des blocs" est le plus lisible dans la première épreuve, le style "*alternatives emboîtées If-Not-End*" est le plus lisible dans la seconde épreuve. Les programmes écrit dans les styles "*alternatives emboîtées*", sont présentés de façon indentée, c'est à dire que la disposition spatiale des écritures marque de façon hiérarchique les différentes parties de l'écriture conditionnelle ; les programmes écrits dans le style "*GOTO*" ne sont pas indentés, bien qu'ils pourraient l'être ; selon l'auteur, si les programmes écrits dans le style "*GOTO*" étaient également indentés, cela n'améliorerait pas les performances, car le mode de collecte de l'information est différent dans les deux styles :

- la collecte de l'information dans le style "*GOTO*" s'apparente à **un jeu de piste** ("treasure-hunting", course au trésor), où chaque étape dépend du résultat obtenu à l'étape précédente ;
- la collecte de l'information dans le style avec *alternatives emboîtées* s'apparente au «*shopping*» (ce terme est utilisé également en Français courant : le "shopping" consiste à parcourir un certain nombre de magasins, pour y faire ses achats, sans plan pré-établi). Ainsi, le lecteur d'un programme rédigé dans le style avec *alternatives emboîtées* parcourt de façon linéaire le programme et "met dans son sac" une information lorsqu'elle répond aux conditions imposées.

2.3 Les modes de fonctionnement mental associés aux styles de programmation

La problématique de T.R.G. GREEN concerne des programmeurs professionnels confrontés au choix d'un style de programmation; elle s'insère dans les débats, contemporains de l'article, autour de l'adoption de langages "structurés" tels que PASCAL, au détriment de langages utilisant le style *GOTO*. Notre problématique concerne les premiers apprentissages en programmation, et donc la constitution chez les sujets confrontés à ces apprentissages, de représentations mentales. Nous sommes donc plus intéressé par les modes de fonctionnement mental que repère T.R.G. GREEN que par les performances de lisibilité qu'il mesure. On a deux modes de lecture d'un programme, chacun de ces modes étant étroitement dépendant des structures du langage de programmation. Dans le premier mode, correspondant au "*style GOTO*", on met en oeuvre des **procédures mentales** correspondant à l'exécution par le dispositif, sans nécessairement qu'il y ait compréhension de la structure du programme. L'autre mode utilise des **représentations schématiques** par lesquelles le sujet intègre la structure (réelle ou supposée) du programme.

Ci-dessous, nous précisons la façon dont le choix d'un langage détermine un style

de rédaction des conditionnelles, les conséquences de ce choix dans les premiers apprentissages, puis nous examinons l'intervention des opérateurs booléens pour la simplification des écritures conditionnelles.

2.3.1 L'évolution des langages, et le choix d'un style pour les premiers apprentissages

L'**assembleur**, en tant que code directement interprétable par le processeur, utilise, pour les conditionnelles, des instructions de branchement conditionnées par la valeur d'un indicateur ou "drapeau", en général imposée par l'instruction qui précède.

Exemple :

```
boucle:
  DEC A
  JNZ boucle
```

L'instruction `DEC A` soustrait 1 à l'accumulateur `A`, et positionne l'indicateur de Zéro, selon la valeur prise par `A` (valeur 0 : indicateur positionné). Si l'indicateur n'est pas positionné, l'instruction `JNZ boucle` produit un branchement à l'instruction repérée par l'étiquette `boucle:`.

Le style "*GOTO*" est donc inséparable de la programmation en **assembleur**. Le premier langage symbolique, FORTRAN, traite également les conditionnelles à l'aide de branchements : l'instruction : `IF (400.0-Y),4,8,8` produit un branchement à la ligne 4 si `(400.0-Y)<0`, à la ligne 8 si `(400.0-Y) = 0` et à la ligne 8 si `(400.0-Y)>0` . [HOROWITZ 83][47].Il en est de même de langages pour débutants tels que LSE et BASIC où le programme est structuré en lignes : les branches d'une alternative peuvent être constituées d'instructions, à condition que ces instructions tiennent dans la ligne ; cette limitation conduit à ce que, dès que les actions sont complexes, les branches de l'alternative sont constituées de branchements. Le langage LSE permet un parenthésage des blocs uniquement à l'intérieur d'une ligne, de façon à éviter les ambiguïtés signalées ci-dessus dans les alternatives imbriquées. Nous retiendrons que, pour ces trois langages, le style "*GOTO*" est celui qui s'impose.

Le parenthésage des blocs est présent dans les langages issus de ALGOL, tels que PASCAL, C et aussi dans LISP. L'usage de ce parenthésage ne se réduit pas aux alternatives, mais permet également de parenthéser des conditionnelles plus générales (itératives), ou les définitions de procédure ou de fonctions.

On peut rencontrer également des langages sans parenthésage de blocs, disposant de marqueurs spécifiques pour les alternatives du type "**IF-NOT-END**" ; c'est le cas par exemple de versions "structurées" de BASIC telle que Turbo-BASIC de Borland, ou BASICA de Microsoft ; souhaitant décourager le recours au branchement *GOTO*, mais voulant conserver l'organisation en lignes du programme, et ne pas introduire de parenthésage de bloc, les concepteurs ont ajouté au langage un avertisseur de fin d'alternative. Le langage de programmation du gestionnaire de fichiers DBASE offre la même structure.

```
IF <condition>  THEN
                .....
                ELSE
                .....
ENDIF
```

2.3.2 Le choix d'un style pour l'initiation à la programmation ; conséquences possibles pour les apprentissages

Comme nous l'avons vu ci-dessus, le style est très directement lié au choix du langage. Les principes de programmation modulaire et structurée conduisent à éviter les branchements (*GOTO*) ; ils dirigent donc le choix de l'enseignant le plus souvent vers ce que T.R.G. GREEN appelle le style "*avec alternatives emboîtées*". L'intérêt de l'étude de T.R.G. GREEN est de montrer que chacun des styles conduit à un mode spécifique de prise de connaissance de l'information ("jeu de piste" dans le cas du style "*GOTO*", "shopping" dans le cas du style "*avec alternatives emboîtées*"). Or ces modes sont eux-mêmes liés chacun à un mode d'interprétation du programme : le style "*jeu*

[47] HOROWITZ E. (1983) *Fundamentals of programming languages* Springer Verlag

de piste" correspond à un mode "**dynamique**" qui privilégie la succession des instructions dans **l'exécution du programme**; les styles *avec alternatives emboîtées* imposent un mode "**logique**", où la **conformité du texte du programme au jeu de relations logiques** impliqué par la tâche à programmer est prise en compte de façon prioritaire, pour .

Correspondant à des fonctionnements mentaux différents (l'un basé sur des procédures, l'autre sur des représentations schématiques), ces deux modes d'interprétation d'un programme ne sont pas nécessairement présents simultanément, et les transferts et mise en relation peuvent ne pas se faire: nous avons constaté, par exemple, que des enseignants en stage de formation approfondie à l'informatique, ayant reçu une formation solide en algorithmique, et pratiquant PASCAL éprouvent des difficultés à construire des conditionnelles simples et efficaces en **assembleur**; ils cherchent systématiquement à appliquer un schéma d'alternative complète (`IF ... THEN ... ELSE ...`) au lieu d'utiliser le "saut d'évitement" si naturel en **assembleur**, et dans les langages utilisant *GOTO*. L'élimination du "*GOTO*" se justifie au niveau universitaire et professionnel pour la lisibilité et la structuration du programme. Elle semble conduire à évacuer la dimension temporelle d'exécution de l'interprétation du programme, ce qui, aux niveau des premiers apprentissages, peut réduire le champ des représentations opératoires à disposition du programmeur.

2.3.3 Alternatives et expressions booléennes

Au détour d'un exemple, T.R.G.GREEN note que «*(des) difficultés peuvent être évitées par le simple moyen d'expressions booléennes*». L'acquisition des types de données est un des centres d'intérêt de cette thèse, en particulier nous étudierons en partie IV l'acquisition des booléens; c'est pourquoi nous approfondissons ci-dessous ce point de vue. Montrons par deux exemples que les opérateurs sur les booléens (connecteurs propositionnels de l'algèbre de BOOLE), peuvent permettre de simplifier les écritures conditionnelles, et donc d'éviter à la fois le style "*GOTO*" et le style "*avec alternatives emboîtées*". : si `A` et `B` sont deux expressions booléennes, les programmes suivants sont équivalents, du point de vue du résultat :

```
programme a1 :                  programme a2
IF A and B THEN action1         IF A THEN
           ELSE action2                BEGIN
                                         IF B THEN action1 ELSE action2
                                       END
                                      ELSE action2
```

Ces programmes peuvent différer du point de vue de l'exécution, car si `A` a la valeur "Faux", `B` n'est jamais évalué dans la seconde expression , alors qu'il l'est dans la première[48] . Supposons par exemple que `f` soit une fonction dont l'argument est un entier positif, alors `IF (a>0) and f(a)= b ...` conduira à une erreur si `a` est négatif, alors que `IF (a>0) THEN IF f(a)= b` ne produira pas d'erreur. Avec les mêmes restrictions, les programmes suivants sont équivalents :

```
programme b1 :                  programme b2 :
IF A or B     THEN action 1     IF A   THEN
          ELSE action2                  action1
                                      ELSE
                                        IF B THEN  action1
                                             ELSE  action2
```

Cette discussion ne s'applique pas au style *GOTO* ; en effet, dans les deux exemples ci-dessus, l'emploi du style *GOTO* permet d'éviter le double appel à `action2` (resp. `action1`), sans pour autant employer les connecteurs booléens autres que la négation.

---

[48] Il peut se faire cependant que des dispositions aient été prises dans la conception du langage de programmation pour que dans l'évaluation de l'expression `A and B`, `B` ne soit pas évalué dans le cas où `A` est faux ; [HOROWITZ 83] désigne par le qualificatif «*court-circuit*» ce mode d'évaluation ; [ARSAC 83] généralise ce type de court-circuit à une classe de fonctions à deux arguments sous le nom d'«*appel par nécessité*». Dans le cas de ce mode d'évaluation, les programmes a1 et a 2 sont identiques dans leur exécution.

```
programme a3 :                  programme b3 :
IF (not A) THEN GOTO L2         IF A THEN GOTO L1
L1: IF B THEN GOTO L3           IF B THEN GOTO L1
L2: action2                     action2 GOTO L2
GOTO L4                         L1: action1
L3: action1                     L4:...
L4:...
```

Un autre exemple montre que l'emploi judicieux des booléens peut conduire loin dans la simplification des écritures conditionnelles : [49][JENSEN WIRTH 78] note, à propos des instructions conditionnelles : «*si* `egal` *est une variable de type booléen, un abus fréquent de l'instruction* `"if"` *conduit à écrire* : `if a=b then egal:=true else egal:=false` *alors qu'il suffit d'écrire:* `egal:= a=b`.» Nous proposons donc un troisième "style d'écriture des conditionnelles": celui où les quantités booléennes interviennent pour simplifier les expressions conditionnelles en chaîne. Dans ce style, on fait intervenir la codification des *conditions* sous la forme de variables booléennes, et par conséquent les fonctionnements mentaux liés aux traitements, que considère T.R.G. GREEN, ne suffisent plus: le sujet doit faire intervenir une représentation mentale des données.

2.4 Conclusion

Le texte de T.R.G. GREEN complète utilement l'apport théorique de J.M. HOC puisqu'il permet d'apercevoir dans un cas précis des modes de prise de connaissance de l'information, et d'anticipation de la réponse du dispositif. Dans le mode associé au style *GOTO,* le sujet "fait fonctionner" un mécanisme mental qui, un peu comme un train parcourant un réseau de chemin de fer, se décrit sous la forme d'une suite ordonnée dans le temps de branchements en divers points du programme. Au contraire, dans le mode associé au style avec alternatives imbriquées, ce mécanisme mental est constitué par une lecture linéaire avec collecte d'informations pertinentes. Nous avons montré que le mode d'interprétation du langage est très dépendant du langage, et que le passage d'un mode de fonctionnement mental à l'autre est loin d'être immédiat. La discussion nous a également conduit à montrer que, avec la possibilité d'employer des opérateur booléens, le langage s'enrichit, et les expressions conditionnelles peuvent être simplifiées, mais il est probable que, dans le même temps, l'anticipation ne peut plus s'appuyer seulement sur une représentation du traitement: elle doit faire intervenir une représentation mentale des données. Nous étudierons en partie IV comment le type booléen peut intervenir dans la programmation en concurrence avec la structure alternative, et la façon dont des élèves intègrent ce type booléen dans leurs conceptions.

3. Le profit des "micro-mondes" dans l'apprentissage de la programmation sur ordinateur (E. COHORS-FRESENBORG)

Nous étudions dans ce paragraphe un texte rapportant une analyse des processus d'acquisition chez les débutants en informatique, et un travail d'ingénierie didactique. Ce texte est assez éloigné de cette thèse, en ce qui concerne les concepts en jeu, mais il nous a semblé intéressant de l'étudier car il représente une direction de recherche et d'expérimentation assez divergente de ce qui se pratique dans l'enseignement français. En effet, aussi bien dans les expérimentations autour de LOGO à l'école élémentaire, que dans l'option informatique au lycée, et dans l'enseignement universitaire, les acquisitions sont visées d'emblée dans un langage évolué, de façon relativement indépendante d'un dispositif qui aurait une réalité concrète. C'est évidemment le cas de l'option informatique des lycées qui a pour objectif de rendre les élèves capables d'effectuer une analyse préalable pour une programmation qui se fait elle-même dans un langage de haut niveau (nous présenterons au chapitre 5 cet enseignement). C'est également le cas de l'utilisation de LOGO à l'école élémentaire, puisque les acquisitions

[49][JENSEN K. WIRTH N. (1978) *PASCAL User Manual and Report* Springer Verlag (Traduction française: *Pascal, manuel de l'utilisateur* Eyrolles 1980

se font dans ce langage qui, dans ses principes, favorise l'écriture de définitions plus que l'organisation d'actions, et où le transfert de valeur s'opère par l'emploi de paramètres formels. E. COHORS-FRESENBORG propose au contraire d'appuyer l'acquisition des concepts de la programmation à partir de tâches de programmation de dispositifs effectuant des actions élémentaires avec un langage restreint.

3.1 Une stratégie pédagogique s'appuyant sur un dispositif entièrement descriptible

Pour l'auteur, la construction d'algorithmes par l'élève suppose qu'il dispose d'une «*structure conceptuelle*» dans laquelle il puisse «*implémenter ses idées*» ; on rapprochera cette structure de la notion de «*Système de représentation et de traitement*» (**S.R.T.**) introduite par J.M. HOC. Cette structure conceptuelle, doit correspondre à un dispositif (au sens de J.M. HOC) facilement descriptible en tant qu'unité matérielle capable d'effectuer des tâches élémentaires, et en même temps «*complet*» en ce sens que les concepts fondamentaux de la programmation peuvent y être représentés.

E.COHORS-FRESENBORG fait le choix d'un dispositif connu sous le nom de «*Machine à Registres*». Ce dispositif comporte un nombre fini de registres, mémorisant chacun un entier naturel ; le langage comporte, pour chaque registre, les instructions d'**incrémentation** (ajouter 1) et de **décrémentation** (soustraire 1), ainsi que l'itération contrôlée par le contenu du registre : "**tant que contenu non nul ...**". Il est possible de construire pour ce dispositif tout algorithme correspondant à une fonction calculable. Selon l'auteur, le cadre conceptuel correspondant permet de *faire fonctionner des notions* telles que les **algorithmes**, la **complexité des algorithmes**, la **grammaire** d'un langage, la notion de **fonction calculable**. Le "curriculum" proposé par l'auteur consiste à faire progresser conjointement l'acquisition des notions d'algorithme et de fonction calculable ; cependant, du fait du choix de la machine à registres comme dispositif, les fonctions construites par les élèves se limitent aux fonctions sur les entiers naturels. Les exemples donnés par l'auteur appartiennent au domaine des fonctions polynomiales.

Remarques sur le langage :Le langage du dispositif n'inclue pas le transfert d'une valeur d'un registre à un autre (qui serait une première forme de l'affectation). Le transfert de valeur, s'il se révèle nécessaire dans un problème, doit être programmé (donc construit par les élèves) par exemple sous la forme suivante :

```
Transfert de R1 vers R2   (utilise le registre 3) :
Tant que R2 > 0  décrémente R2
Tant que R1 > 0  décrémente R1 incrémente R2 incrémente R3
Tant que R3 > 0  décrémente  R3 incrémente R1
```

La notion de variable est présente sous une forme assez primitive : lorsqu'un problème nécessite l'utilisation d'un registre pour des calculs intermédiaires, le choix de ce registre parmi les registres non utilisés, est arbitraire ; ce choix est donc laissé à l'interpréteur du langage, le programmeur se contentant de désigner ce registre par une lettre. La conséquence est que les élèves utilisent des identificateurs qui sont en fait des paramètres locaux de sous-programmes ; ainsi, l'identificateur H peut être utilisée dans le programme ci-dessus, à la place du registre 3 :

```
Transfert de R1 vers R2 (utilise le paramètre local H):
Tant que R2 > 0  décrémente R2
Tant que R1 > 0  décrémente R1 incrémente R2 incrémente H
Tant que R3 > 0  décrémente H incrémente R1
```

On notera, par différence, que les schémas d'apprentissage couramment pratiqués dans l'enseignement français, et que nous avons cité plus haut, considèrent les variables et l'affectation comme des notions "déjà construites"[50] (puisque présentes dans le

[50] ou, «*pré-construites*», selon le terme employé par [CHEVALLARD 85]. Y. CHEVALLARD note que la «*pré-construction*» des notions s'établit par le croisement d'énoncés du langage et de situations «*sur-déterminées*», et en déduit que ces notions «*pré-construites*» sont nécessairement peu résistantes à des changements de cadre de référence (elles ne sont pas décontextualisables). C'est ce que nous vérifierons pour les variables et l'affectation, du fait du

langage) pour lesquelles l'élève doit découvrir une *signification* à partir des exemples qui lui sont présentés.

3.2. Les micros-mondes

Dans la perspective d'établir un curriculum pour l'enseignement secondaire, E. COHORS-FRESENBORG appuie l'acquisition de la «*structure conceptuelle*» sur des «*représentations*» : il s'agit de dispositifs matériels, appelés par l'auteur «*micro-mondes*» qui réalisent à divers niveaux la structure de "machine à registre"[51] :

- le premier de ces micro-mondes est une boîte séparée en casiers pouvant contenir des allumettes, et matérialisant de cette façon les registres ; l'opérateur réalise lui-même l'incrémentation et la décrémentation des registres ; il ne peut connaître le contenu d'une boîte, sauf si elle est vide. L'élève construit des algorithmes qu'il exécute pour des données arbitraires, mais il n'y a pas de concrétisation observable de ces algorithmes, en dehors de l'action elle-même.
- le second dispositif est constitué d'éléments d'un réseau ferroviaire miniature ; des briques "compteurs" peuvent être incrémentées ou décrémentées au passage du train, et peuvent basculer un aiguillage pour la valeur 0. Dans ce micro-monde, la construction d'un algorithme se concrétise par la réalisation d'un réseau. On dispose alors d'un résultat observable, en fait proche d'un "organigramme" (*flow-chart*).
- le troisième dispositif est un véritable langage implanté sur ordinateur. Les actions permises (incrémentation, décrémentation), sont indiquées par une lettre indexée sur le numéro du registre concerné. Le programme est organisé en lignes ; un numéro en début de ligne indique que les actions doivent être itérées tant que le registre de numéro correspondant n'est pas nul. Les sous-programmes sont nommés par une lettre, suivie des registres paramètres.

Selon E.COHORS-FRESENBORG, les "micro-mondes" permettent des expressions différentes pour un même problème ou un même algorithme, et les différents passages d'un micro-monde à l'autre aident à la formation des concepts chez les élèves. Nous rapprochons ce point de vue de la notion de «*cadre*» introduite par R. DOUADY en didactique des mathématiques : «*Deux cadres peuvent comporter les mêmes objets et différer par les images mentales et la problématique développée. Le changement de cadre est un moyen d'obtenir des formulations différentes d'un problème qui sans être nécessairement équivalentes, permettent un nouvel accès aux difficultés rencontrées et la mise en oeuvre d'outils et de techniques qui ne s'imposaient pas dans la première formulation.*» [DOUADY 86][52].

3.3. les résultats expérimentaux: stratégies de résolution et structures cognitives

Une expérimentation a été conduite auprès de 80 élèves du début du secondaire (13 ans) ; l'observation porte sur la résolution de problèmes par les élèves concernant l'écriture de programmes pour la machine à registres avec deux modalités distinctes. La première modalité consiste en une **construction de programmes** : une fonction (par exemple l'addition de deux registres) étant donnée, construire un programme réalisant cette fonction. La seconde modalité est **l'interprétation de programmes** : un programme est donné à l'élève pour analyse ; il doit indiquer la fonction que réalise le programme, ou calculer le nombre de pas de programme en fonction des données (fonction de complexité).

Le premier résultat est l'indépendance des compétences de construction d'une part, et d'interprétation d'autre part. L'étude permet également la mise en évidence de stratégies de résolution distinctes : dans la stratégie «*conceptuelle*», un élève

---

choix de présentation de ces notions dans l'option informatique des lycées.

[51] Bien sûr, ces "représentations" matérielles doivent être distinguées des "représentations" mentales qui s'organisent en S.R.T.

[52] DOUADY R. (1986) Jeux de cadres et dialectique outil-objet" *Recherches en didactique des Mathématiques*, Vol 7.2. Editions La Pensée Sauvage

commence par une analyse du problème, puis le structure, et tente de construire un cadre conceptuel ; dans la stratégie «*séquentielle*», les élèves tentent une première solution avant d'avoir complètement structuré leurs idées ; le développement de leurs idées passe par un dialogue avec le matériel.

D'autres études expérimentales rapportées dans l'article de E. COHORS-FRESENBORG conduisent à distinguer deux types de *structure cognitive* ; le premier type est constitué par des **relations** , le second s'exprime en termes de **fonctionnement**. Un élève dont la *structure cognitive* ne prend pas en compte le fonctionnement d'une machine éprouve par exemple, des difficultés à réparer une erreur dans un réseau. On rapprochera ces deux types de *structures cognitives* des modes d'interprétation des écritures conditionnelles présentées par Th GREEN : le mode d'interprétation jeu de piste correspondant à une structure cognitive privilégiant le **fonctionnement**, et le mode «*shopping*» correspondant à une structure privilégiant les **relations**. Il est important de noter que le type de structure cognitive présent chez un élève n'implique pas le choix par cet élève d'une stratégie de résolution : des élèves adoptant une stratégie «*conceptuelle*» peuvent préférer représenter les algorithmes dans le micro-monde des réseaux ferroviaires, ce qui indique que leur structure cognitive est du type «*fonctionnement*".

3.4. Discussion

3.4.1 Une variable didactique pertinente: construction des concepts ou recherche de significations.

La notion de «*structure conceptuelle*» apporte un éclairage supplémentaire à la notion de **S.R.T.** introduite par J.M. HOC. L'idée d'aider à constituer cette structure conceptuelle par le choix d'une machine dont le langage a des capacités d'expression limitées, et qui est donc facilement descriptible et représentable nous paraît conduire à l'introduction d'une variable didactique pertinente : dans la progression mise en place par E. COHORS-FRESENBORG l'acquisition d'éléments conceptuels tels que l'affectation et les variables est recherchée à travers l'élaboration par les élèves de procédures comme solution locale dans des problèmes. Dans la pratique de l'enseignement de l'informatique en France au contraire, ces éléments sont présents d'emblée dans le langage proposé aux élèves : l'explicitation a priori du langage n'est pas possible, car elle supposerait que les élèves aient acquis les éléments conceptuels qui y sont présents ; les élèves doivent donc faire émerger une signification pour chacun des éléments de ce langage, à partir des exemples, et des activités de programmation qui lui sont proposées.

Nous proposons donc de distinguer deux stratégies pour l'acquisition des concepts. Dans l'une, le concept nouvellement introduit serait le produit de l'intégration de procédures construites pour un dispositif dont l'élève possède une représentation mentale adéquate, à l'occasion de la résolution de problèmes. Ces procédures se verraient reconnaître leur portée générale et seraient institutionnalisées par l'enseignant ; on enrichirait ainsi le dispositif en même temps que le Système de Représentation et de Traitement de l'élève s'enrichit. Dans l'autre stratégie, les éléments conceptuels dont l'acquisition est recherchée seraient présents dans le langage comme formes syntaxiques: les élèves ne disposeraient donc pas au départ d'un système de représentation et de traitement adéquat, et le but de l'enseignement serait alors de faire évoluer ce système, la résolution de problèmes variés conduisant l'élève à élaborer des significations pour les concepts. Nous ne sommes évidemment pas en mesure de préciser dans quels domaines une stratégie serait plus pertinente que l'autre, et nous verrons ci-dessous que le choix d'une stratégie peut résulter de contingences propres au contexte d'enseignement.

Parmi les résultats cités par E. COHORS-FRESENBORG, nous retenons qu'il n'y pas de lien de causalité entre le type de *structure conceptuelle* présent chez un élève (s'exprimant en terme de **relations** ou au contraire en terme de **fonctionnement**) et la *stratégie* (**conceptuelle** ou **séquentielle**) qu'il met en oeuvre. Ceci est important à prendre en compte pour l'enseignant qui serait tenté d'inférer à partir de la stratégie de résolution de l'élève (la plus facile à observer), son type de fonctionnement cognitif, (moins facile à mettre en évidence), voire même de penser faire évoluer le

fonctionnement cognitif de l'élève en l'amenant à modifier sa stratégie de résolution.

3.4.2 Une transposition problématique au cadre expérimental de notre recherche

Dans le contexte de l'option informatique des lycées (notre terrain de recherche, que nous présenterons au chapitre 4) l'ingénierie didactique exposée par E. COHORS-FRESENBORG semble difficile à transposer : les élèves viennent en option informatique d'abord pour apprendre à utiliser la puissance de l'ordinateur. En conséquence, la limitation de cette puissance ou l'utilisation de dispositifs disposant d'un langage aux possibilités d'expression plus limitées ont de fortes chances de ne pas être acceptées.

D'autre part, l'idée de construire une représentation mentale permettant le fonctionnement des concepts dont l'apprentissage est visé, à partir d'un dispositif dont le langage soit rapide à apprendre, et qui soit facilement matérialisable dans plusieurs cadres n'est pas directement transposable à d'autres objets que les entiers naturels. Si l'on considère les chaînes de caractères, qui tiennent une place importante dans notre thèse, il faudrait une machine qui possède non seulement des registres, mais aussi une mémoire ; une chaîne serait représentée par un couple de registres : un pointeur sur la mémoire (pour le premier caractère de la chaîne), et un registre pour la longueur. Il faudrait également une instruction de transfert de mémoire à mémoire (pour une telle description des chaînes de caractères, voir le chapitre 5). On se rend compte que cette construction est plus lourde et moins facilement matérialisable qu'une machine à registres. Une ingénierie didactique basée sur cette construction serait cependant intéressante à tenter : nous considérons qu'elle sort du cadre de cette thèse qui vise d'abord à repérer des difficultés d'acquisition spécifiques aux différents types de données, et à examiner comment, dans un cadre didactique s'éloignant peu du cadre habituel les énoncés des problèmes posés aux élèves peuvent leur permettre de rencontrer et de surmonter ces difficultés. Les éléments conceptuels généraux tels que les variables ou l'affectation auront été rencontrés par les élèves à l'occasion d'autres activités, mais l'acquisition de ces concepts sera seulement amorcée ; il s'agira non seulement de continuer cette acquisition, mais aussi de faire émerger une signification pour ces concepts dans le nouveau cadre introduit avec des types différents de ceux que les élèves ont rencontré jusque là. Cependant, dans l'esprit de la stratégie d'apprentissage adoptée par E. COHORS-FRESENBORG, nous gardons comme perspective de présenter aux élèves la structure de donnée à étudier à partir d'éléments minimaux, constitués de fonctions simples, de façon à permettre aux élèves de construire les autres éléments de la structure comme outils pour la résolution d'un problème (voir par exemple, chapitre 5, le cas de la fonction `position` pour la structure "chaîne de caractères").

3.4.3 L'utilisation de cadres

Dans le travail de E. COHORS-FRESENBORG, l'expression des algorithmes sur plusieurs unités matérielles ayant chacune ses contraintes spécifiques contribue à faciliter les acquisitions. Cette utilisation paraît également difficile à transposer aux premiers apprentissages s'appuyant sur un langage de programmation évolué. On peut penser à utiliser la comparaison entre les expressions dans des langages différents, ou à comparer l'expression d'une même structure algorithmique portant sur des données de types différents, mais ceci sort du champ des premiers apprentissages.

4. Une étude didactique : signification et fonctionnement du concept de variable informatique chez des élèves débutants (R. SAMURCAY)

4.1 Un problématique proche de celle de cette thèse

Le texte que nous analysons dans ce quatrième paragraphe constitue une étude didactique dont la problématique est proche de celle de cette thèse : en effet, elle porte sur les processus d'acquisition des concepts, avec comme terrain expérimental l'option informatique. Elle diffère de celle des études que nous avons présenté dans les paragraphes 1 et 2 par le public qui constitue le terrain expérimental, et le domaine

conceptuel étudié. Elle diffère également de celle du texte de E. COHORS-FRESENBORG, en ce sens qu'il s'agit d'une observation à un instant donné, et non de la constitution d'un curriculum ; il nous semble également que dans le contexte de l'option informatique, où se place l'étude de R. SAMURCAY l'acquisition des concepts est recherchée à travers l'émergence chez l'élève de *significations* pour les éléments du programme, ce programme étant écrit en langage évolué, alors que E. COHORS-FRESENBORG fait dépendre l'acquisition des concepts de l'explicitation d'un *schéma de fonctionnement* d'un dispositif suffisamment simple pour que cette explicitation soit possible.

4.2 Les concepts de base

Pour R. SAMURCAY, les concepts de boucle, de variable, et d'affectation sont indissociables. C'est à travers la planification par le sujet du traitement des variables dans une itération que l'on peut observer significativement la formation, chez le sujet, de ces concepts. Cette planification se divise en trois tâches liées entre elles :

- la mise à jour (par une affectation à l'intérieur du corps de boucle ; nécessité de construire une régle d'évolution de ces variables)
- la construction du test d'arrêt : variables sur lesquelles porte ce test ; place dans le corps de boucle.
- l'initialisation des variables itératives.

4.3 L'étude expérimentale

Elle a pour objectif de repérer les conceptions des élèves de Seconde après une quinzaine de séances, en différenciant selon leur difficulté cognitive les différentes tâches (déclaration, initialisation, mise à jour, intervention dans la condition d'arrêt) et en classant les variables selon la facilité que les élèves ont à se les représenter (variable compteur plus facile à se représenter que les variables d'accumulation "récapitulatives"). Il est demandé à l'élève de compléter des programmes dont la fonction leur est indiquée : dans une question, ils doivent compléter la partie "initialisation " pour la variable compteur et deux variables "multiplicatives" ; dans une seconde question ils doivent élaborer le test d'arrêt ; dans la troisième question, il doivent construire la mise à jour des variables dans le corps de boucle.

On remarquera que la méthode d'observation diffère des méthodes employées par les auteurs observés ci-dessus. Chez ces auteurs, on trouve soit des observations de sujets en situation de construction de programme (HOC, COHORS-FRESENBORG), soit des observations de sujets réalisant des tâches d'interprétation de programmes (GREEN, COHORS-FRESENBORG). E. COHORS-FRESENBORG donne d'ailleurs comme résultat l'indépendance des capacités de construction et d'interprétation, mais sans préciser davantage les liens entre ces capacités. R. SAMURCAY propose des programmes à compléter qui impliquent donc à la fois une tâche d'interprétation et une tâche de construction. Il est certainement difficile de dire actuellement dans quelle mesure les performances d'un sujet à une tâche de complétion permettent de se prononcer sur ses compétences concernant une tâche de construction complète. Nous considérons cependant qu'une telle modalité d'observation est bien adaptée à l'observations de débutants, pour l'essentiel dans l'incapacité de mener à bien de façon autonome une tâche de construction. L'intérêt des résultats rapportés ci-dessous confirme la pertinence de la méthode.

4.4 Les résultats

1 les élèves réalisent mieux l'affectation de la variable `COMPTEUR` que d'autres variables itératives ; il réalisent préférentiellement l'initialisation par une lecture (entrée d'une valeur par l'utilisateur) ou par une affectation de la valeur 0.

2 les élèves ont bien intégré les contraintes syntaxiques concernant les marqueurs d'itération (`UNTIL`, pour marquer la fin d'une itération commencée avec `REPEAT`). Mais l'écriture de la condition d'arrêt témoigne dans plus de la moitié des cas de fausses représentations concernant la capacité de contrôle du dispositif : la condition d'arrêt est alors une expression par laquelle l'élève signifie que le résultat attendu est obtenu ; cette expression peut avoir le statut informatique de condition

(égalité de deux variables, par exemple), et témoigne alors de confusions sur la signification des variables ; ce peut être aussi une expression sans valeur de condition, par exemple un identificateur de variable, ou une expression à valeur numérique : dans ce cas, l'expression peut être interprétée comme la traduction mot à mot de l'idée de l'élève ; par exemple, devant compléter un programme où le produit de x et y est calculé itérativement dans une variable d'identificateur RESULTAT, certains élèves écrivent: REPEAT...UNTIL x*y , ou REPEAT... UNTIL RESULTAT.

3 dans une tâche de complétion du corps de boucle , l'erreur principalement rencontrée est la non évolution des variables itératives (compteur et résultat) : à chaque "tour de boucle", la valeur initiale est reprise, et la valeur du résultat (obtenue au "tour de boucle" précédent) est perdue. L'itération est conçue par ces élèves comme la répétition d'une action, indépendamment des valeurs prises par les variables. S'agissant par exemple d'écrire un programme qui "multiplie par 3 le carré d'un nombre un certain nombre de fois", les élèves écriront :

```
repeat   result:= nombre * nombre   result:=result*3 until…
```

ou même sans itération :

```
result := (3*(nombre*nombre))*nbrecoup
```

R.SAMURCAY note : «*on peut estimer que l'instruction de changement de valeur de la variable d'accumulation demande une représentation fonctionnelle particulière de la relation décrite par l'énoncé*». Les représentations fonctionnelles que les élèves peuvent avoir rencontrées en mathématiques (par exemple, la représentation de la puissance d'exposant n d'un nombre a comme produit de n nombres égaux à a) sont trop éloignées de la suite des calculs à opérer, et surtout ne prennent pas en compte les états successifs des variables.

4.5 Discussion

4.5.1 Représentation et signification

Au sein des représentations mentales que se font les sujets, R. SAMURCAY insiste sur la *signification* du concept de variable. Le contenu sémantique que le programmeur, qu'il soit débutant ou expérimenté, attache soit à la notion de variable en général, soit à telle variable du programme conduit en effet à ce que sa représentation ne soit pas un schéma traduisant de façon mécanique la façon dont les variables interviennent dans le langage du dispositif : la signification que le programmeur débutant attache par exemple à la variable "COMPTEUR" dans une itération ne se réduit pas à ses propriétés, elle se rattache aux situations , informatiques ou non, où le sujet a rencontré une entité voisine; la signification d'autres variables (variables d'accumulation, en particulier), peut être plus difficile à dégager à cause du manque de situations d'où le sujet pourrait faire émerger des éléments sémantiques voisins. La dimension des significations est plus nettement présente dans le texte de R. SAMURCAY que dans les études analysées précédemment. Au début de ce sous-chapitre, nous avons attribué au contexte de l'option informatique cette importance prise par les significations dans le schéma d'acquisition ; nous avons également souligné dans les conclusions du sous-chapitre 3 qu'il y là une variable didactique : privilégier l'explicitation d'un schéma de fonctionnement d'un dispositif suffisamment simple pour que cette explicitation soit possible, et penser que des concepts tels que celui de variable se dégageront à partir de l'intégration de procédures élaborées par les élèves, ou présenter d'emblée ces concepts, et penser que leur signification se dégagera à partir des exemples que les élèves auront à traiter.

4.5.2 Le type des variables, et le contexte sémantique impliqué par ce type

Cette question n'est pas envisagée dans l'étude de R. SAMURCAY. Les algorithmes sur lesquels portent l'étude expérimentale sont du domaine du calcul arithmétique (exponentiation...). Un des résultats est que les élèves ne parviennent pas à passer du mode de relation fonctionnelle impliqué par les écritures mathématiques, à celui qui serait nécessaire pour l'écriture des algorithmes. Pour reprendre les termes de J.M. HOC, il y a là deux **S.R.T.** entre lesquels les élèves ne parviennent pas à établir des codes. Ces deux **S.R.T.** sont éloignés en ce qui concerne les relations fonctionnelles, ce qui crée les difficultés, mais ils sont proches du point de vue du symbolisme, ce qui

peut créer l'illusion qu'il n'y a pas de difficulté.

On peut donc s'interroger sur la spécificité des difficultés rencontrées dans l'acquisition d'un concept comme celui de variable selon le type des variables, et le contexte sémantique impliqué par ce type. On débouche ainsi sur les questions suivantes : y a t'il une variabilité de la signification du concept avec le type ; par exemple le transfert de valeur opéré par l'affectation prend t'il des significations différentes selon le type, selon le domaine sémantique attaché à ce type ; en quoi des affectations telles que `a←2*a` (a étant une variable numérique), `a←a+b` (a et b étant des variables chaîne) , `a← not( rep = 'oui')` (a étant une variable booléenne) ont elles des similitudes et des différences du point de vue de l'interprétation qu'en font les élèves ; dans quelle mesure la fréquentation de variables de plusieurs types facilite t'elle l'émergence chez l'élève d'une signification du concept ?

4.5.3 Signification des variables dans un contexte itératif, et signification dans un contexte non-itératif

Dans l'étude de R. SAMURCAY, les difficultés sont repérées à l'occasion d'exercices portant sur l'itération. A juste titre, l'auteur note que le concept de variable prend totalement son sens seulement avec l'itération : une entité dont la valeur évolue sans que sa signification change. Le problème est que, en plus des difficultés liées à la compréhension du concept de variable, la conception d'une itération recèle de nombreuses autres difficultés : construction de la condition d'arrêt, articulation des instructions dans le corps de boucle avec la condition d'arrêt et l'initialisation des variables. Il nous semble donc que si l'on ne peut s'appuyer sur des éléments de signification construits antérieurement à l'itération, l'accumulation des difficultés conduira de nombreux élèves à l'échec. La question posée est donc celle des rapports entre la signification des variables que les élèves peuvent construire par l'écriture de programmes non-itératifs, et celle qui leur serait nécessaire pour construire des itérations.

L'étude de R. SAMURCAY montre que les élèves se construisent des éléments de signification du concept de variable liés au contexte itératif, et en quelque sorte indépendants d'une conception plus générale qui pourrait fonctionner aussi bien dans un contexte non-itératif : par exemple, l'initialisation systématique à la valeur 0 des variables d'accumulation est révélatrice d'une conception installée spécifiquement pour l'itération : les variables doivent être mises à zéro avant de commencer. Le petit nombre des séances qui précédent l'épreuve passée par les élèves explique sans doute que le lien n'a pu être fait entre les significations construites dans le cadre itératif et le cadre non-itératif.

Il nous semble important de nous poser la question de ce lien : sur quels éléments de signification acquis dans un cadre non-itératif, peut-on s'appuyer pour construire une signification adaptée à l'écriture de programme itératifs ? Y a t'il des conceptions construites à l'occasion de l'écriture de programmes non-itératifs qui se révèlent des obstacles à un fonctionnement correct du concept de variable pour l'écriture de programme itératifs ?

5. Résumé des conclusions

Du texte de J.M. HOC, nous retenons d'abord les éléments théoriques : les dispositifs sont représentatifs d'une unité agissant sur l'environnement selon des régles entièrement énonçables, et sont organisés hiérarchiquement selon la relation d'abstraction ; les Systèmes de Représentation et de Traitement (**S.R.T.**) du sujet, lui permettent de se représenter un tâche effectuable sur un dispositif donné, et de faire fonctionner mentalement ce dispositif ; la résolution d'un problème de programmation conduit le sujet à élaborer des codes lui permettant de passer d'un **S.R.T.** où il peut se représenter une solution à un **S.R.T.** correspondant au dispositif.

Nous retenons également les 3 degrés d'intériorisation du langage du dispositif : au premier degré, le sujet ne dispose pas d'un **S.R.T.** lui permettant de se représenter

la solution d'un problème de programmation sous la forme d'un algorithme ; au second degré, une telle représentation est possible, mais le **S.R.T.** dans laquelle elle s'exprime est trop éloigné d'une représentation d'un dispositif programmable pour que la solution soit traduite dans le langage du dispositif. Au troisième degré, le sujet dispose de **S.R.T.** organisés hiérarchiquement, lui permettant d'envisager une «*solution abrégée*» directement compatible avec le dispositif, de résoudre à part les modules pour lesquels il ne dispose pas d'emblée d'une solution, et de traiter en premier lieu les données de façon abstraite, de façon à ce que les parties les plus difficiles guident le choix de la représentation des données. Nous avons noté que les méthodes de programmation, telles que la programmation structurée, ou l'abstraction des données s'adressent en fait à des sujets parvenus à ce troisième degré d'intériorisation du langage du dispositif. Concernant les débutants, la question essentielle est plutôt la façon dont cet ensemble de **S.R.T.** organisés hiérarchiquement peut se construire.

De l'étude de T.R.G GREEN, nous retenons l'existence de modes de fonctionnement mental spécifiques à des styles de programmation, et la difficulté que peut représenter pour le sujet le passage d'un mode à l'autre. Nous avons examiné comment les langages de programmation actuellement disponibles conduisent à un style de programmation, et les conséquences du choix d'un langage pour l'initiation. Nous avons indiqué qu'un travail sur les variables de type booléen nous semble de nature à simplifier l'écriture des expressions conditionnelles, et donc à réduire les difficultés d'interprétation de ces écritures, étudiées par T.R.G. GREEN.

Le curriculum présenté par E. COHORS-FRESENBORG nous paraît une tentative pour asseoir l'apprentissage de la programmation sur un système de représentation appelé «*structure conceptuelle*", représentatif d'un dispositif facile à décrire pour les élèves, et susceptible de plusieurs concrétisations matérielles. Nous avons noté que cette approche privilégie les représentations du dispositif en termes de fonctionnement. Des éléments conceptuels, comme ceux de variables, ou d'affectation ne sont pas présents de façon primitive dans le langage ; si les élèves peuvent se les construire en tant qu'outils, la question se pose de savoir comment ils pourraient leur attribuer une signification dépassant une simple représentation en tant que schéma de fonctionnement, de façon à généraliser leurs acquis à des situations de programmation moins spécifiques. Nous avons également indiqué que cette approche est difficilement transférable à l'étude que nous envisageons dans cette thèse, à cause du terrain expérimental, et du domaine conceptuel que nous projetons d'étudier.

Dans le texte de R.SAMURCAY, nous relevons l'importance des significations et de leur articulation avec les représentations en terme de fonctionnement : les élèves traitent correctement la variable COMPTEUR dans une itération parce qu'ils lui donnent une signification claire, et qu'ils relient cette signification à la façon dont l'ordinateur fait évoluer sa valeur. Par contre, d'autres variables itératives ont une signification moins claire pour les élèves, ou plus précisément, ils ne peuvent déduire de la signification qu'ont ces variables en mathématiques, une signification qui serait adapté au traitement par un dispositif informatique. A la suite de l'analyse du texte de R.SAMURCAY, nous posons les questions suivantes, en lien avec l'étude que nous projetons dans cette thèse : la signification que les élèves attachent aux variables et aux concepts associés (affectation , itération) dépend t'il du type des données ? Sur quels éléments de signification acquis dans un cadre non-itératif, peut-on s'appuyer pour construire une signification adaptée à l'écriture de programme itératifs ?

Chapitre 3 Cadre général de recherche

à quel public s'intéresser?
le sujet confronté à l'activité de programmation
le problème de programmation

Ce chapitre présente le cadre général d'hypothèses dans lequel cette recherche s'insère. Ce cadre prend en compte l'analyse des contenus d'enseignement du chapitre 1, et l'analyse de travaux de psychologie de la programmation et de didactique de l'informatique du chapitre 2. Il tente de répondre aux premières questions assez générales, qu'on se pose en début de recherche: une investigation au niveau des débutants étant imposée par l'état actuel de la recherche, il nous faut faire des hypothèses sur le fonctionnement cognitif de ce public face à l'activité de programmation, discuter ce qu'est, à ce niveau, un problème de programmation faisant intervenir les concepts de base de l'informatique et compatible avec les langages accessibles aux débutants.

1. Dans l'état actuel de la didactique de l'informatique, la recherche doit s'intéresser prioritairement à un public hétérogène sans finalité professionnelle

Le choix d'un niveau d'enseignement de l'informatique auquel s'intéresser est la première question que doit se poser un chercheur en didactique tentant de mieux comprendre le fonctionnement cognitif de sujets confrontés aux apprentissages en informatique. Dans l'état actuel de l'enseignement de cette discipline, de nombreux cursus s'adressent à des publics assez spécifiques: étudiants persévérant en informatique dans le cadre d'un enseignement à finalité professionnelle (études professionnelles d'informatique, écoles spécialisées) ou étudiants abordant l'informatique comme complément à une solide formation de base dans le domaine scientifique (écoles d'ingénieur, premiers cycles scientifiques des universités, seconds cycles universitaires d'informatique). La didactique de l'informatique a fait l'objet, jusqu'ici, de peu de recherches et donc, dans une observation portant sur ces publics si particuliers, la part des spécificités des étudiants et de l'enseignement risque d'être difficile à faire. Pour commencer à préciser un cadre d'analyse général, il vaut mieux partir de l'observation d'un public d'élèves n'ayant pas ces spécificités: il sera plus facile de situer ensuite, dans ce cadre général, les particularités de publics spécifiques[53] . C'est pourquoi nous avons fait le choix d'observer des élèves non sélectionnés pour leurs acquisitions antérieures, dans le cadre d'un enseignement sans vocation professionnelle. L'option informatique des lycées que nous présenterons au chapitre 4, nous a paru un terrain d'observation adapté, surtout en première année (classe de Seconde), où le public est le plus hétérogène. Ce travail se situe donc dans une perspective proche de travaux tels que [SAMURCAY 85] et [COHORS-FRESENBORG 87], plutôt que de travaux de psychologie de la programmation tels que [GREEN 77] et [HOC 77]. Néanmoins, l'ensemble de ces travaux, présenté au chapitre **2** va nous être

[53] Au niveau de publics spécifiques, une question souvent posée est celle de l'acquisition de capacités à structurer un programme par une analyse préalable. Il nous semble difficile, dans l'état actuel des recherches en didactique de l'informatique, de savoir si les réticences ou difficultés des étudiants à mener une telle structuration sont liées à des conceptions propres à cette structuration, ou se rapportent à des conceptions erronées plus profondes des objets de base de l'informatique. Il nous semble que de nombreuses études proposant ou discutant des stratégies pédagogiques dans ce domaine manquent d'une étude préalable situant les conceptions et les connaissances des étudiants concernés. Voir par exemple «Une initiation à l'analyse descendante» C. GEOFFROY Actes du premier colloque de didactique de l'Informatique éditions EPI [GEOFFROY 88].

utile pour préciser un cadre général d'analyse des conceptions du sujet face à une tâche de programmation.

2. Le sujet confronté à l'activité de programmation: représentations du dispositif et relation aux concepts

2.1 Les dispositifs informatiques; représentations matérielles, traitements et données

- De [HOC 77], nous avons retenu que les **dispositifs**, qu'ils existent réellement ou qu'ils soient virtuels, sont représentatifs d'unités matérielles agissant sur l'environnement selon des régles entièrement énonçables, et sont organisés selon des hiérarchies d'abstraction: par exemple, un langage de programmation "*évolué*" comme le sous-ensemble de PASCAL utilisé par les élèves observés dans [SAMURCAY 85] diffère de la "*machine à registre*" présentée dans [COHORS-FRESENBORG 87] en ce sens que des éléments du langage (affectation, opérations arithmétiques...) sont primitifs dans le sous-ensemble de PASCAL alors qu'ils doivent être *construits* dans le langage de la "machine à registre". De même le sous-ensemble de PASCAL contient des "avertisseurs d'itération" permettant de composer un calcul itératif à partir de sa définition, sans se préoccuper de la façon dont le dispositif gère les ruptures de séquentialité[54], alors que dans le langage de la "machine à registre", le programmeur dispose seulement d'une forme élémentaire de conditionnelle de sortie d'itération et doit donc chercher une adaptation de la définition du calcul itératif compatible avec les capacités d'expression du langage de la "machine à registre".
- Un dispositif est susceptible de "**représentations matérielles**" diverses: la relation entre le dispositif et sa "représentation" matérielle peut être entièrement explicite dans le cas de la "machine à registre" (représentation à l'aide de boîtes d'allumettes, de circuits de chemin de fer...); elle l'est beaucoup moins dans le cas du sous-ensemble de PASCAL, celui ci étant relié à une unité matérielle par un ensemble hiérarchisé de couches logicielles (langage machine, système d'exploitation, micro-code du processeur..) qui ne se laisse pas décrire simplement.
- Dans le dispositif informatique, nous distinguerons d'une part la **structuration des données**, et d'autre part les **traitements** que le dispositif réalise sur ces données. Nous avons vu au chapitre 1 que les notions liées à la structuration des données se sont dégagées progressivement au cours de la jeune histoire de l'informatique, alors que les notions liés au traitements (théorie des algorithmes...) sont présentes dès le départ de l'informatique. Une structuration des données se caractérise comme un "**type abstrait**", c'est-à-dire comme un système formel (une construction axiomatique) faisant intervenir des fonctions sur les objets considérés et leurs propriétés. La "**représentation**" d'une structure dans une autre est un lien abstrait entre les types, analogue à la "*construction* " d'une structure axiomatique dans une autre en mathématiques.
- les structures de données seraient seulement des structures axiomatiques si elles n'étaient pas associées à des traitements dans la **résolution de problèmes de programmation**: la résolution des ces problèmes fait progresser de pair la construction d'algorithmes et la structuration des données, celles ci étant considérées alternativement comme types abstraits et à l'aide de leurs représentations dans d'autres structures.

2.2 Les S.R.T. comme structures mentales locales: éléments relatifs aux données et éléments relatifs aux traitements

Nous avons rencontré dans [HOC 77] les **Systèmes de Représentation et de Traitement (S.R.T.)**: un **S.R.T.** est le produit, chez le sujet, de l'intériorisation d'un domaine de tâches: il permet à un sujet de se représenter un tâche donnée

[54] En fait, ceci n'est possible, dans ce langage que pour une certaine classe d'itérations. Nous approfondirons cette question au chapitre 5.

effectuable sur un dispositif donné, et de faire fonctionner mentalement ce dispositif pour cette tâche. Les **S.R.T.** d'un sujet sont organisés hiérarchiquement selon une relation d'abstraction, et la résolution d'un problème de programmation consiste pour le sujet à élaborer des codes lui permettant de passer d'un **S.R.T.** où il peut se représenter une solution, à un **S.R.T.** correspondant au dispositif; il y a donc, au cours de l'activité de résolution, création par l'élève de codes mentaux reliant différents **S.R.T.**

Un **S.R.T.** donné chez un sujet donné est donc pour nous la structure mentale qui lui permet à un moment précis d'une tâche précise de construire une solution ou de vérifier sa validité. Les conduites que l'on peut observer chez le sujet découlent du **S.R.T.** qu'il fait fonctionner, mais, plus qu'un ensemble de régles de conduite, le **S.R.T.** est une véritable "machine mentale", que le sujet peut faire fonctionner sans que, bien sûr, les régles de fonctionnement soient explicitées ni entièrement explicitables. Le(s) **S.R.T.** que le sujet met à contribution pour la résolution de problèmes de programmation présentent une organisation analogue à celle dispositif pour lequel il programme. C'est pourquoi un **S.R.T.** correspondant à un dispositif informatique contiendra de façon organisée, d'une part certains éléments propres aux **objets** eux-mêmes, et d'autre part des éléments relatifs aux **traitements**.

- Dans le domaine des **objets**, le **S.R.T.** utilisé par le sujet lui permet, par exemple, de distinguer ou non des objets d'un type particulier, d'intégrer ou non la variabilité des objets, la possibilité de les nommer, de faire changer leur valeur; de disposer ou non d'un langage sur ces objets permettant d'exprimer et de combiner les fonctions propres à ces objets. Nous dirons que ces éléments s'organisent, au sein du **S.R.T.**, **en plan(s) relatif(s) aux objets**[55]. Quand, par exemple, un sujet confronté à la complétion d'un dessin en "géométrie-tortue", doit calculer l'angle de rotation nécessaire pour "positionner" la tortue dans la direction qui lui permettra de terminer le dessin, il prend en compte la position atteinte, l'orientation de la tortue, les propriétés géométriques des dessins élémentaires... Il met donc en oeuvre des éléments de représentation mentale de la tortue et du dessin élémentaire à réaliser et des procédures mentales, qui constituent pour nous le plan relatif à l'objet "tortue" du **S.R.T.** que le sujet utilise.
- Le **S.R.T.** permet d'autre part au sujet de se représenter, de planifier, de faire fonctionner mentalement des **traitements complexes**, et donc d'articuler les actions élémentaires sur les objets. Les *plan(s)* relatif(s) aux traitements dans un **S.R.T.** comprennent des représentations des capacités de traitement (réelles ou supposées) du dispositif, des significations que le sujet attribue aux divers éléments syntaxiques marquant le traitement, des procédures d'exécution mentale des traitements complexes... En reprenant l'exemple ci-dessus, le sujet confronté à la complétion d'un dessin en "géométrie-tortue" ne doit pas seulement imaginer comme nous l'avons vu, des actions élémentaires comme les rotations de la tortue, et ses déplacements. En s'appuyant sur sa conceptions de l'objet "tortue", il doit également trouver dans son **S.R.T.** les éléments pour imaginer et coordonner les déplacements de la tortue: selon ses capacités, il trouvera seulement des représentations et des procédures mentales directement déduites de l'expérience sensible (par exemple, il pourra se représenter le tracé d'un carré comme une suite d'actions simples non organisées en répétition), ou il pourra intégrer des représentations plus ou moins adéquates de processus répétitifs avec contrôle, il pourra penser ces processus en termes d'exécution (suite ou répétition d'action) ou en termes de transformation de situations (situation initiale, situations intermédiaires, situation finale..).

Les dispositifs les plus abstraits (par exemple celui que constitue un langage de programmation "évolué"), permettent en premier lieu de considérer toute manipulation sur des objets comme un *calcul* (c'est à dire un processus `données→ résultats`, entièrement explicitable à l'aide de règles formelles) et nous avons vu au chapitre 1 que ceci conduit à considérer d'une part une définition abstraite des objets, et d'autre

[55] En fait, ces conceptions s'organisent en sous-systèmes du **S.R.T.** relatifs chacun à un type de données.

part des représentations qui relient entre elles les structures abstraites. Ces dispositifs comprennent également des traitements généraux indépendants des objets sur lesquels ils opèrent: c'est ainsi que par exemple, la concaténation d'un caractère à une chaîne est un calcul de même nature que l'addition de deux nombres, et le "retournement d'une chaîne de caractères" est une itération isomorphe à l'exponentiation obtenue comme multiplication itérée. Au contraire, dans le **S.R.T.** d'un sujet confronté à une tâche nouvelle de programmation, les éléments relatifs aux objets peuvent rester marqués par une conception *pré-calculatoire* (par exemple, la concaténation peut être intégrée comme un déplacement effectif de caractères), ou les représentations peuvent ne pas être distinguées des définitions abstraites (par exemple, un nombre pourra ne pas être distingué de la chaîne de ses chiffres en base 10) et les éléments relatifs aux traitements peuvent être conçus de façon très dépendante des objets sur lesquels ils portent, ce qui se traduit, par exemple, par le fait que le sujet peut ne pas transférer d'un domaine d'objets à l'autre ses capacités algorithmiques. Nous verrons au §3.1 par quels mécanismes se constitue un **S.R.T.** de ce type.

2.3 Le niveau des concepts informatiques; conséquences pour la stratégie d'apprentissage

Les **S.R.T.** ont été introduits par J.M. HOC pour rendre compte des conduites du sujet face à un ensemble varié de tâches, allant de la conduite de dispositifs matériels à l'utilisation de langages de programmation évolués. L'enseignement de l'informatique, ne vise pas la simple maîtrise des tâches, mais, à travers cette maîtrise, la possibilité de raisonner au niveau des concepts informatiques, et plus seulement au niveau des domaines de tâches. Nous considérerons que la disponibilité pour un sujet d'un concept informatique, se traduit en premier lieu par des capacités de transfert, de différenciation, de coordination d'un **S.R.T.** à l'autre; ainsi, la disponibilité du concept de variable permettra à un sujet de faire le lien entre l'affectation dans un langage impératif comme PASCAL, et l'inscription d'une donnée ou d'une formule dans une cellule d'un tableur comme **Multiplan**, de distinguer une affectation initiale, faite une seule fois dans un programme, d'une affectation se répétant dans le corps d'une itération. En tant qu'exemple de raisonnement au niveau des concepts, l'étude de la complexité d'un algorithme de tri suppose de s'abstraire des objets sur lequel opère l'algorithme, et de la règle de comparaison qui fonde le tri: la complexité prend en compte le nombre d'affectations et/ou de comparaisons effectuées, indépendamment des objets sur lesquels porte l'affectation et indépendamment du type de comparaison. Le niveau des concepts permet donc aussi au sujet de raisonner sur un algorithme ou une structure de données indépendamment du contexte dans lequel ils ont été construit. En particulier, et par différence avec le niveau d'un **S.R.T.** élaboré pour une tâche de programmation particulière, le niveau des concepts est celui où le sujet peut raisonner sur la structuration des données indépendamment d'une structure algorithmique, et sur la structure algorithmique indépendamment de la structuration des données.

Comment un sujet peut-il accéder au niveau des concepts informatiques de base? Mener une étude didactique auprès de débutants, suppose que le chercheur annonce ses hypothèses en matière de stratégie pédagogique. Nous considérons d'abord que le champ des concepts de base en informatique est constitué de notions liées fortement (notion de variable, de traitement conditionnel, d'itération, de structuration des données...), au point qu'il n'est guère possible de définir l'une sans les autres et que par conséquent, des stratégies pédagogiques qui viseraient à présenter ces notions a priori et de façon séparée videraient ces concepts de leur sens, réduisant l'enseignement à l'apprentissage d'un langage. De même, une présentation des algorithmes indépendamment des données ou l'inverse, empêcherait, que ces notions prennent leur sens. Il est donc nécessaire, suivant l'analyse de [BROUSSEAU 86], déjà citée au chapitre 1, que l'enseignant se livre à un travail de *recontextualisation* , «*inverse* » de celui du savant : «*il doit produire une* ***recontextualisation*** *et une* ***repersonnalisation*** *des connaissances*», de façon à ce que les connaissances deviennent «*la connaissance de l'élève, c'est-à-dire une réponse assez naturelle à des conditions (...)indispensables pour qu'elles aient un sens pour lui*». Le travail des

élèves est de «***redécontextualiser*** *et* ***redépersonnaliser*** *leur savoir, (...) de façon (à l') identifier avec le savoir qui a cours dans la communauté scientifique et culturelle de leur époque*». Comme nous l'avons vu au chapitre 1, la (courte) tradition de l'enseignement de l'informatique opère la recontextualisation des concepts par le choix de "**problèmes de programmation**". Un "problème de programmation" se définit comme l'énoncé d'une tâche qu'il s'agit de faire réaliser à un dispositif informatique; de façon générale, le travail de l'élève consiste donc, à partir d'une machine universelle, à réaliser une machine adaptée à une tâche donnée. Mais bien sûr, comme dans un problème de mathématiques, la façon d'obtenir la solution, la discussion des solutions possibles est plus importante que la solution elle-même; nous avons vu au chapitre 1, avec [ARSAC 83] et [PAIR MOHR SCHOTT 88], des exemples d'une telle recontextualisation.. Nous en déduisons que, au niveau des premiers apprentissage, une stratégie pédagogique doit viser en premier lieu la constitution de **S.R.T.** locaux, construits en réponse à des problèmes de programmation où les concepts fondamentaux interagissent, permettre ensuite la mise en relation et la structuration de ces **S.R.T.**, et proposer des situations conduisant les étudiants à considérer ces concepts en dehors des contextes.

Notre recherche portant sur les premiers apprentissages, son sujet principal est par conséquent l'étude de la constitution de ces **S.R.T.**, et il en résulte que le choix des types de problèmes proposés aux étudiants, en particulier la façon dont les concepts sont mis en jeu dans ces problèmes, est fondamental. Nous discutons ce choix au paragraphe suivant.

3. A partir de quels problèmes de programmation observer et faire évoluer les représentations mentales des élèves ?

En fonction des hypothèses énoncées ci-dessus concernant le fonctionnement mental du sujet confronté aux apprentissages en informatique, nous présentons d'abord un schéma de l'activité de programmation que nous pensons adapté au débutant, puis nous discutons plusieurs conceptions des problèmes de programmation, à partir d'un choix d'auteurs, et, enfin, nous exposons les principes qui nous guident dans la conception des problèmes dont nous étudierons la résolution aux chapitres suivants.

3.1 Les problèmes de programmation dans les premiers apprentissages et la constitution des S.R.T.

La figure ci-dessous décrit l'activité de programmation telle que nous la concevons dans le cadre de l'alphabétisation informatique.

<table>
<tr><th colspan="2">Activité de Programmation</th><th>Sujet débutant en Programmation</th></tr>
<tr><td>Connaissances données</td><td>≈≈≈≈≈≈≈≈ Connaissances résultats</td><td>Situation connue (familière)</td></tr>
<tr><td>↓ codification ↓</td><td>↑interprétation↑</td><td></td></tr>
<tr><td>Informations données</td><td>⟶ Informations résultats
traitement algorithmique</td><td>Spécificités, contraintes, rencontrées et intégrées progressivement.</td></tr>
</table>

En partie gauche de la figure, un schéma de l'activité de programmation tiré de [ARSAC

86][56]. Ce schéma nous paraît bien décrire la résolution de problèmes simples, et le fonctionnement de débutants qui sont très marqués par une conception intuitive (naïve) des objets et des traitements. [ARSAC 86] introduit ce schéma pour présenter les implications culturelles qui pourraient résulter selon lui d'une alphabétisation massive en informatique: la résolution d'un problème de programmation suppose que le sujet se représente des "*connaissances données*" et des "*connaissances résultats*"; le plus souvent, elles sont issues de sa connaissance sensible des "objets du monde"; la résolution suppose également que le sujet imagine un lien entre ces connaissances, ce lien pouvant être un "traitement à la main", c'est à dire d'un traitement dans les objets du monde, ou même un mode de résolution ne faisant pas appel à un traitement conscient (par exemple, repérer le caractère du milieu dans une chaîne de longueur impaire...): nous pouvons résumer la première ligne du tableau en disant que le sujet dispose d'un **S.R.T.** intuitif lui permettant de se représenter le problème, **S.R.T.** qui est influencé par son expérience sensible, qui est adapté à un traitement "dans les objets du monde" (en particulier l'élève dispose dans ce **S.R.T.** de *plans* au sens de [HOC 87]), mais pas à un traitement sur le dispositif informatique. La "*codification*" résulte d'un certain nombre de conventions "réductrices" du point de vue du sens, permettant d'associer aux connaissances données des entités informatiques; l'"*interprétation* " est l'opération inverse. Les données codifiées sont considérées pour leurs propriétés formelles, et non à partir d'un contexte intuitif; d'autre part, pour être adaptées à un traitement algorithmique, les données codifiées présentent des spécificités par rapport aux données intuitives (par exemple, une chaîne de caractères doit être indexée par un ordinal, une liste permet l'accès seulement à son premier élément et au reste, un booléen a seulement deux valeurs). Le raisonnement sur les données *codifiées* est donc d'une autre nature que le raisonnement sur les objets intuitifs, et par conséquent, pour produire une solution (un traitement pour un dispositif informatique donné) le sujet doit disposer d'un **S.R.T.** distinct du **S.R.T.** du problème intuitif, lui permettant de se représenter les données sous leur forme codifiée, et un traitement sur ces données.

Il y a problème de programmation si le sujet ne dispose pas d'un **S.R.T.** lui permettant de se représenter les données codifiées, et un traitement sur ces données. La résolution du problème est en fait la constitution de ce **S.R.T.**. Schématiquement, on peut distinguer deux cas de constitution de ce **S.R.T.**:

- le sujet ne peut s'appuyer sur des représentations mentales déjà constituées: dans ce cas, il doit déduire la solution en opérant des différenciations à partir du **S.R.T.** intuitif au fur et à mesure de la découverte des contraintes et spécificités du dispositif. Les conditions dans lesquelles s'opèrent ces différenciations sont certainement très diverses, dépendantes des connaissances antérieures des élèves, aussi bien que des caractéristiques du dispositif pour lequel les élèves programment. Par exemple, un dispositif de bas niveau, comme la *machine à registres* présentée dans [COHORS-FRESENBORG 87] (voir chapitre 2) comporte un ensemble de régles facilement énonçables et est susceptible de représentations matérielles; les fonctions de base (affectation, jeu d'instructions arithmétique...) devant être construites par le programmeur, les représentations mentales se construisent parallèlement. Au contraire, dans le cas d'un sous-ensemble d'un langage évolué (que nous avons rencontré au chapitre 2 avec la présentation de [SAMURCAY 85]), l'énoncé a priori des régles de formation serait sans intérêt, et, en tant que machine, ce sous-ensemble ne possède pas de représentation matérielle simple[57]. Dans ce cas, le sujet devra donc se constituer des représentations mentales opératoires à l'aide des exemples proposés par l'enseignant, à l'aide de

---

[56] J. ARSAC "L'informatique et le sens. Une gigantesque mutation culturelle ?" in Actes du colloque du CREIS histoire et épistémologie de l'informatique Mai 86.

[57] Des représentations matérielles sont parfois employées ; elles sont locales et métaphoriques, c'est-à-dire qu'elles illustrent une partie seulement du langage, et ne doivent pas être prises au pied de la lettre : par exemple, la métaphore de la pile d'assiettes pour la pile `LIFO` ne prend pas en compte le fait que dans `empiler (a)` la valeur de a n'est pas modifiée, alors qu'elle l'est dans `dépiler (a)`.

connaissances issues d'apprentissages dans d'autres disciplines, à l'aide des "effets en retour" du dispositif...[58].

- le **S.R.T.** adapté à un traitement algorithmique peut être constitué à l'aide d'éléments présents dans d'autres **S.R.T.**. Dans ce cas, le sujet doit opérer des relations et des différenciations entre **S.R.T.** Par exemple, dans un problème impliquant la représentation d'un graphe orienté (parcours sur un plan...), le sujet disposera d'un (de) **S.R.T.**(s) correspondant à un traitement sur un tableau numérique, et d'un (de) **S.R.T.**(s) comprenant des calculs sur les booléens. Il devra en quelque sorte "fédérer" ces **S.R.T.**, remettre en cause certains de leurs aspects trop liés aux contextes dans lesquels ils ont été construits, établir des relations avec le **S.R.T.** intuitif du problème.

En fait, si le problème s'adresse à des débutants, et s'il ne s'agit pas de la traduction directe d'un procédé de calcul vu dans les apprentissages antérieurs, le mode le plus courant sera celui décrit dans [HOC 87] et rapporté au chapitre 2 (§1.3): face à une tâche de programmation consistant à modéliser une situation qu'il connait, le sujet débutant ne trouve pas dans le **S.R.T.** correspondant au dispositif pour lequel il programme, un "*plan*" lui permettant d'imaginer une solution; il fait donc appel à des *plans* d'un **S.R.T.** servant dans la situation familière.

Ce mode de constitution éclaire les caractéristiques des **S.R.T.** dons nous avons parlé au §2.2. Le plus souvent, les **S.R.T.** familiers n'intègrent pas la notion de calcul sur les objets, et les traitements y sont liés aux objets: par exemple, dans un **S.R.T.** familier correspondant à un traitement sur les chaînes, les caractères sont repérés par des dispositions spatiales, et le traitement opère sur ces dispositions: ce n'est pas un calcul, contrairement au traitement des nombres dans le **S.R.T.** mis en oeuvre pour l'exponentiation. En construisant sa solution à partir d'un plan conçu pour la situation familière, le sujet **intègre dans son S.R.T. correspondant au dispositif informatique, des caractéristiques des représentations et traitements qui lui servent dans cette situation, et ceci d'autant plus largement que ces caractéristiques n'empêchent pas la résolution**: ainsi, par exemple, programmant le parcours d'une chaîne de caractères, le sujet pourra construire une itération fonctionnant effectivement tout en ayant une compréhension des éléments en jeu largement influencée par la situation familière: l'index pourra par exemple, ne pas être compris comme numérique, l'avertisseur d'itération `POUR i ← 1 JUSQU'A longueur(chaîne)` pouvant être compris comme donnant à la variable `i` son statut d'index parcourant la chaîne, et non comme organisant de façon précise l'évolution de `i` comme compteur de boucle.

3.2 Données et traitements dans les premiers apprentissages; discussion de quelques choix concernant les problèmes de programmation

Les hypothèses des paragraphes précédants, en particulier l'interdépendance que nous avons supposée au sein des **S.R.T.** constitués par le sujet en réponse à un problème de programmation, des éléments relatifs aux traitements, et des éléments relatifs aux données (§ 2.2), et le schéma de l'activité de programmation dans les premiers problèmes, fondé sur la différenciation entre le niveau intuitif où est posé le

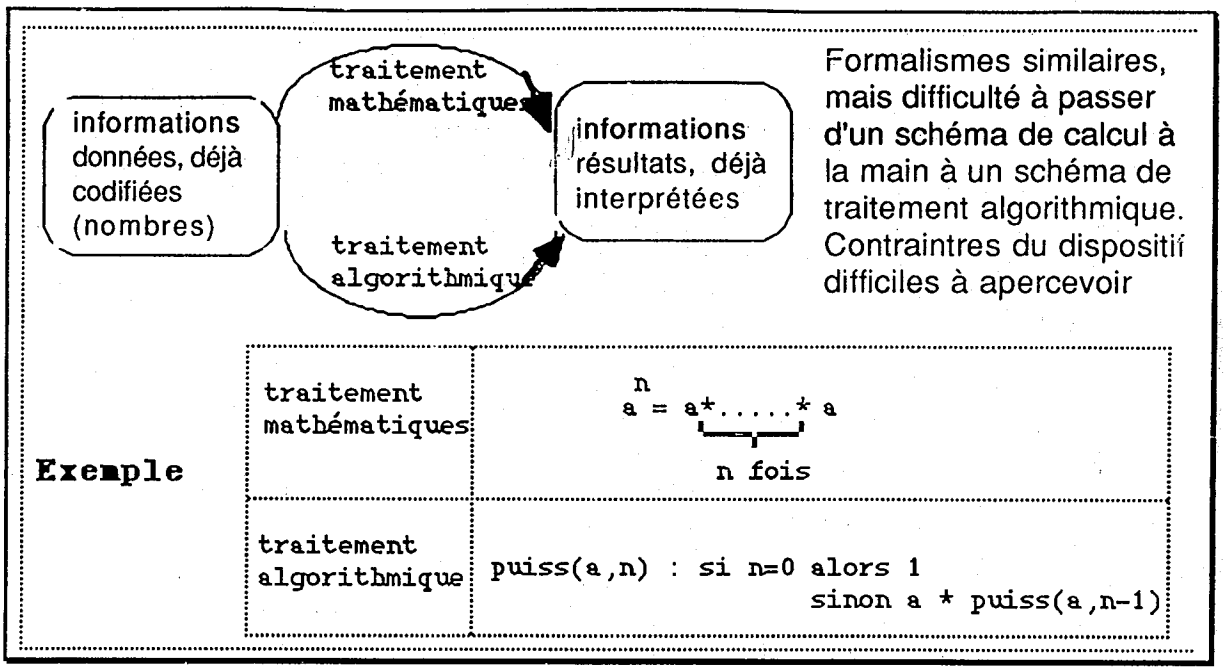
[58][ROGALSKI 87] (J. ROGALSKI «Savoirs et savoir-faire en informatique 1987 ») distingue deux voies possibles de constitution de nouvelles connaissances. Dans une première voie, "Le fonctionnement en lui-même des éléments d'un système inconnu peut permettre la constitution d'une signification des notions et des régles de ce système." J. ROGALSKI fait remarquer que, dans cette voie, le sujet "s'appuie souvent implicitement sur l'existence de systèmes de représentations et de traitements (...) suffisamment voisins pour introduire du sens au-delà des régles d'utilisation", et donc "Une autre voie est la construction de nouvelles connaissances à partir de celles de cadres préexistants avec des précurseurs(...)". Elle note que des précurseurs peuvent avoir un caractère facilitant pour les nouvelles représentations (précurseurs "producteurs"), ou au contraire un caractère inhibant (précurseurs "réducteurs"). Dans cet esprit, nous examinerons au chapitre 5 ce qu'il en est de ces précurseurs pour chacune des notions en jeu dans les apprentissages .

problème, et le niveau des données codifiées que suppose le traitement algorithmique (§ 3.1), nous conduisent à faire l'hypothèse que les problèmes proposés aux débutants seront d'autant plus intéressants que **le sujet pourra déduire du contexte intuitif un(des) plan(s) lui permettant d'amorcer une résolution, tout en imposant, à travers l'adaptation de ce(s) plan(s) pour le dispositif informatique des différenciations par rapport au contexte intuitif, allant dans le sens de l'abstraction des données (données considérées par leurs propriétés, multiplicité des représentations...) et de la reconnaissance du caractère général des traitements algorithmiques.** Pour mieux préciser le type de problème sur lequel nous ferons travailler les élèves dans la partie expérimentale de cette thèse, nous situons en premier lieu quelques "stratégies types" en les discutant par rapport à l'analyse qui précède.

3.2.1 Des problèmes où les premiers traitements portent sur des données "déjà codifiées"

Une stratégie peut consister à viser l'acquisition de capacités algorithmiques en laissant de côté la question de la codification des données, et donc à proposer des problèmes sur des données "déjà codifiées", c'est-à-dire des données numériques dans des problèmes qui se ramènent à la programmation d'algorithmes numériques. [SAMURCAY 85] et [COHORS-FRESENBORG 87], présentés au chapitre 2 s'intéressent à l'acquisition de capacités concernant les traitements sans considérer comme une variable la nature des objets sur lesquels portent ces traitements. Or les objets numériques sur lesquels interviennent les traitements dont ils étudient l'acquisition sont liés à un contexte de calcul numérique algorithmique et ont les particularités de ce contexte: R. SAMURCAY constate par exemple que, face à la programmation d'un algorithme d'exponentiation, les élèves qu'elle observe éprouvent des difficultés à passer d'une représentation de l'exponentiation issue des mathématiques à une représentation permettant la construction d'un traitement adapté au dispositif informatique.

Dans ce type de problème, en effet, le traitement en mathématiques et le traitement algorithmique sont isomorphes (le traitement par la machine peut en effet s'envisager comme une systématisation du "calcul à la main") mais supposent probablement chez le sujet des représentations différentes (représentation du calcul "déjà achevé" en mathématique, représentation du calcul "en train de se faire" en informatique). Plus précisément, en mathématiques, les procédures de calcul sont guidées par une définition statique et par conséquent ne dégagent pas de plans qui soient importables dans un **S.R.T.** informatique: par exemple, la définition de la puissance nième du nombre a comme produit de n nombres égaux à a ne constitue pas un plan de calcul bien que pour des valeurs particulières de a et n, elle permette une multiplicité de calculs. Pour surmonter ce type de difficulté, il faudrait sans doute que le sujet approfondisse à la fois sa conception du traitement mathématique et du traitement informatique, ce qui sort de notre propos. le schéma ci-dessous résume cette discussion.



Un autre écueil d'une stratégie où les algorithmes opèrent sur des données numériques, est que d'autres types interviennent: le type booléen comme type des conditions dans les instructions conditionnelles, les chaînes pour les entrées/sorties... Même si ces types jouent un rôle secondaire dans l'algorithme, et sont donc passés sous silence par l'enseignant, ils peuvent interagir fortement dans la compréhension qu'a le débutant du problème et de sa solution; nous avons montré au chapitre 1 (§ 4.5) que dans un problème de recherche du jour de la semaine où "tombe" le jour de Noël pour un millésime donné ([ARSAC 80]) la question du calcul du résultat sous forme d'une chaîne explicite (`Lundi, Mardi...`) peut, si elle n'est pas résolue, handicaper la recherche de l'algorithme de résolution par le débutant.

3.2.2 Plusieurs univers de programmation pour une méthode de construction d'algorithmes

Dans une autre direction, des auteurs comme [DUFOURD 88][59] ont proposé des "*univers de programmation*", comme situations *"stimulantes"* pour développer des capacités algorithmiques. Un univers de programmation est constitué, pour eux, d'un certains nombre de situations où un même type de donnée peut représenter les objets; ils proposent à titre d'exemple, un "*univers traditionnel*" constitué des booléens, des nombres et des chaînes, un univers graphique (la "*géométrie tortue*") et un "*univers de relation*" (les bases de données). Ils se proposent dans l'article, de montrer comment une même méthode de construction d'algorithmes peut opérer dans ces différents "univers". Par rapport à la stratégie précédente, cette direction a l'avantage de prendre en compte la dimension des "objets", mais, au vu de ce que nous avons discuté au chapitre 1, la conception des objets dans l'approche proposée par ces auteurs semble assez artificielle et séparée de la construction des algorithmes, puisqu'elle repose sur l'idée de situations "naturellement" liées à un type d'objet: il serait possible , selon les auteurs, d'obtenir a-priori une bonne structuration des données dans laquelle opérerait ensuite une méthode de construction d'algorithmes standard (la méthode MEDEE développée dans [DUCRIN 84]). Il n'y a donc pas pour eux, construction jouant de façon conjointe sur l'aspect structuration des données et sur l'aspect traitement, mais bien conception séparée de la structuration des données et de l'algorithme. En l'absence d'étude expérimentale, la façon dont se font les acquisitions des élèves dans ce cadre méthodologique reste hypothétique: les auteurs avancent

[59] [DUFOURD 88] G. et J.F. DUFOURD "Des univers et des outils variés pour commencer à programmer" in Actes du Premier Colloque Francophone de Didactique de l'Informatique Editions EPI

l'idée que les élèves pourraient s'approprier le langage lié aux objets par un fonctionnement "en mode bureau", puis pourraient ensuite appliquer la méthode MEDEE. Si les élèves débutants étaient capables d'appliquer directement une méthode où les concepts liés aux traitements sont d'emblée conçus comme indépendants des objets sur lesquels ils portent, cela voudrait dire que les **S.R.T.** qu'ils mettent en oeuvre comportent des éléments liés aux traitements conçus directement de façon indépendante des données, ce qui infirmerait nos hypothèses. Nous montrerons, à l'aide d'une étude expérimentale au chapitre 7, que ces hypothèses ont une validité et que, par conséquent il est raisonnable de mettre en doute l'efficacité de la méthode indiquée dans l'article.

3.2.3 L'approche de S.PAPERT

D'autres stratégies d'apprentissage font intervenir des environnement de programmation marqués par un type particulier; c'est le cas de la "*géométrie-tortue*" que [PAPERT 80][60] a introduit comme exemple de "*micro-monde*". Dans ce cas, il ne s'agit plus comme ci-dessus, de généraliser à plusieurs domaines de données une méthode de construction d'algorithme, mais bien de promouvoir un domaine particulier de données, en donnant comme raison qu'il serait le plus adapté aux acquisitions recherchées: «*L'arithmétique est une mauvaise introduction à la réflexion heuristique. En revanche, la géométrie Tortue en est une excellente. Grâce à ses qualités de syntonie (syntonie corporelle et syntonie du moi), le simple acte d'apprendre à dessiner à la tortue permet à l'enfant de se forger un modèle d'apprentissage fort éloigné du modèle dissocié*». Le domaine considéré paraît cependant très particulier: en tant que domaine mathématique, il s'agit d'une géométrie faisant largement appel aux mesures de distances et d'angles; en tant que domaine informatique, il fait intervenir un objet complexe constitué en fait comme un ensemble de variables globales non explicites: l'écran de dessin, la position , le cap, l'état du crayon... D'autre part, le domaine conceptuel où les acquisitions sont recherchées n'apparaît pas clairement: il s'agit d'acquérir un langage, donc en principe les acquisitions sont recherchées en informatique, mais il est souligné, en de nombreux points de l'ouvrage de S. PAPERT, que l'informatique n'est qu'un moyen. Des acquisitions en mathématiques seraient en fait recherchées, mais aussi des capacités transversales: acquisition de la pensée scientifique, heuristique...En revenant au domaine qui nous intéresse (les acquisitions en informatique) il se pourrait que des capacités sur un type d'objet aussi particulier soient assez peu transférables sur d'autres objets: [SAMURCAY ROUCHIER 90], suite à une étude expérimentale sur des élèves travaillant à l'écriture de procédures récursives, d'abord dans un environnement graphique, puis dans un environnement constitué de listes numériques, notent que «*le modèle relationnel qui a une valeur opératoire dans un cadre graphique, présente une grande fragilité. Dans un domaine nouveau (suites d'entiers) sa mise en oeuvre rencontre de difficultés importantes.*». En fait, le programmeur travaille, pour les problèmes les plus simples, dans un **S.R.T.** intuitif lié à son "schéma corporel", mais dès que les contraintes deviennent plus fortes, le **S.R.T.** qui serait nécessaire intègre des compétences en mathématique et/ou en informatique. J. HILLEL ([HILLEL 85]) a mené une série de recherches sur ce sujet. Dans le domaine des acquisitions en informatique, il ressort que, dans un environnement de géométrie tortue présentant des contraintes particulières, des élèves de 11 à 13 ans peuvent développer des capacités à construire un parcours de la tortue sous forme de procédures composées, et, dans une moindre mesure, à réutiliser pour d'autres tâches, une procédure écrite antérieurement. Les domaines où le sujet peut construire ces compétences à partir de la seule activité de programmation semblent donc en nombre plus limité que ce qu'avait.imaginé S. PAPERT.

3.3 Notre choix: des problèmes où les objets du monde réel et les données informatiques sont bien différenciés

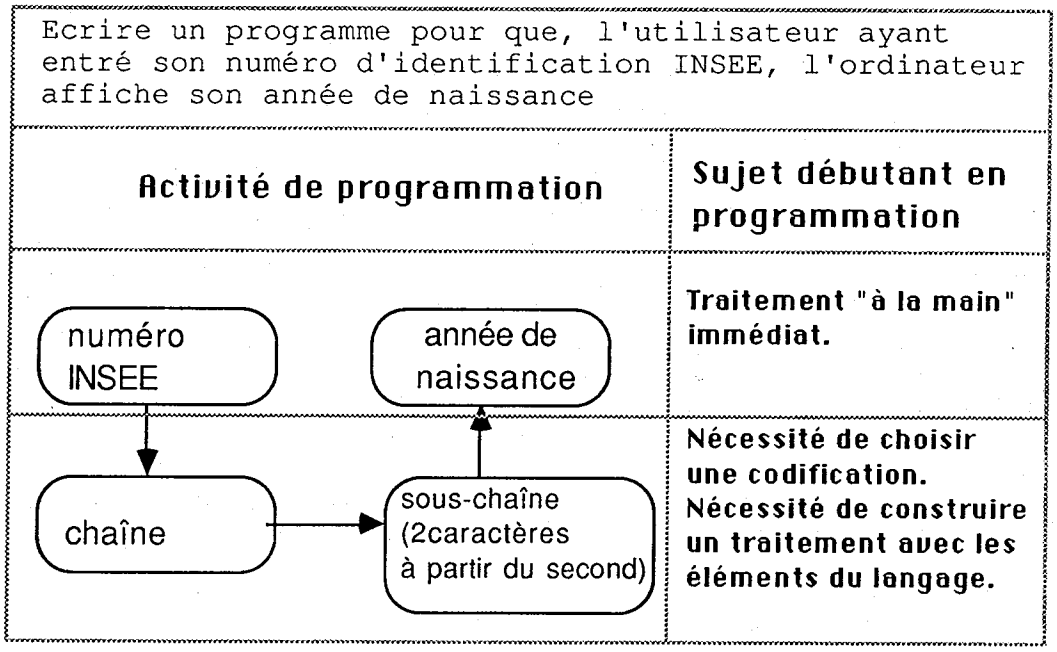
A la suite de l'analyse que nous avons développée en § 3.1, nous pensons

---

[60][PAPERT 80] : S. PAPERT MINDSTORMS Basic Books 1980 Traduit en français sous le titre de Jaillissement de l'esprit Flammarion 1981.

nécessaire de travailler sur des problèmes où la réalisation d'une tâche par un dispositif informatique impose de distinguer nettement le traitement formel sur les objets codifiés, du traitement dans les objets du monde. Il est important que, dans ces problèmes, la codification soit réellement repérable, et donc, d'une part, que le niveau où est posé le problème permette aux élèves de considérer de façon intuitive les objets en jeu et le lien entre données et résultats (et donc de disposer de représentations schématiques d'un traitement dans ces objets, c'est à dire de *plans* susceptibles de constituer un point de départ pour la résolution[61]) et d'autre part que la recherche d'une solution pour le dispositif informatique impose, quant à lui, de considérer ces objets sous forme de données informatiques et le traitement comme un "calcul" (direct ou algorithmique) sur ces données codifiées. De plus, les problèmes doivent faire intervenir de façon liée les concepts informatiques de base et au contraire, éviter de faire intervenir des éléments conceptuels (par exemple mathématiques) en dehors du champ de l'informatique; en effet les questions d'acquisition en informatique sont suffisamment complexes pour ne pas faire intervenir des difficultés d'analyse provenant d'acquisitions différenciées que pourraient avoir les sujets étudiés dans d'autres disciplines.

Nous pensons que les problèmes ainsi définis sont à même de faire rencontrer aux élèves les spécificités du dispositif lorsque la mise en oeuvre d'un *plan* (c'est à dire d'une représentation schématique d'une solution) importé du domaine familier imposera l'adaptation des **S.R.T.** de ce dispositif. Cette nécessaire adaptation entraînera des difficultés pour l'élève dans la construction de son programme, mais nous attendons que leur résolution conduise à l'intégration des contraintes et à une meilleure représentation du dispositif informatique. Nous précisons, au paragraphe suivant, la façon dont il est possible d'utiliser des types pré-définis des langages de programmation pour ces problèmes. Voici, au préalable, un exemple dans la figure suivante:



Le numéro d'identification INSEE est une chaîne de 13 chiffres attribuée en France à chaque résident par l'institut national de la statistique [CHAMBADAL][62]. Le

[61] Nous avons souligné que des problèmes revenant à la programmation d'algorithmes mathématiques ne permettent généralement pas à l'élève d'avoir recours à des plans de ce type.. Il en est de même de problèmes posés directement dans les objets informatique comme par exemple [ANDERSON 84] ANDERSON J.R. FARELL R. SAUERS R. 1984 Learning to program in LISP. *Cognitive Science 8*

[62] [CHAMBADAL 83] L.CHAMBADAL Calcul Pratique HACHETTE 1983

premier chiffre code le sexe (MASCULIN, FEMININ) de la personne, les deux chiffres suivants sont les deux derniers chiffres de l'année de naissance. Cet exemple peut paraître rudimentaire: nous verrons que sa programmation pose un réel problème à des élèves de seconde (partie III). Le traitement "à la main" est en effet immédiat: l'opérateur humain "sépare" les deux chiffres pertinents et les interprète aussitôt en tant qu'année: il n'a pas conscience du type d'entité (numérique ou chaîne de symboles) qu'il considère. Au contraire, pour programmer, il faut commencer par un choix de codification: si l'on choisit de codifier un numéro INSEE comme une donnée numérique, il faudra d'abord s'assurer qu'il existe un type d'entier dans le langage pouvant prendre une valeur à 13 chiffres décimaux, ce qui est rarement le cas. Si c'est le cas il faudra ensuite faire appel à des fonctions arithmétiques (division entière par 10^{10} puis reste modulo 100) Un autre choix est de considérer le numéro INSEE comme une chaîne de chiffres. Les fonctions sur les chaînes couramment présentes dans le langage (que nous présenterons au chapitre 5) permettent le calcul des deux derniers chiffres de l'année sous la forme d'une `sous-chaîne`, qu'il est possible de concaténer à 18 ou 19 pour obtenir l'année de naissance..

3.4 Quelle utilisation des types prédéfinis ?

Pour des débutants, la codification ne peut être séparée d'une représentation à l'aide des types prédéfinis dans le langage réellement utilisé, ce qui implique de raisonner sur cette représentation plutôt que sur des objets abstraits; le choix des types est réduit à ceux qui sont présents dans les langages de programmation les plus courants : types numériques, chaînes de caractères, et booléens. Pour chacun de ces types, nous nous proposons d'examiner la possibilité de construire des problèmes où le traitement sur les objets codifiés à l'aide de ces types se distingue suffisamment du traitement dans les objets du monde.

3.4.1 Les types numériques.

Ils sont conçus pour représenter dans un certain domaine de validité les entités mathématiques correspondantes (nombres entiers, nombres réels). La question du rapport entre les nombres tels que les élèves les conçoivent à la suite de l'enseignement de mathématiques qu'ils ont reçu, et la codification dans le dispositif informatique peut donc leur être posée à partir de problèmes où les entités numériques sortent du domaine de validité pour le langage utilisé : nombres entiers trop grands en valeur absolue, addition de nombres réels dont l'un est très petit par rapport à l'autre.[63]. Mais les réponses sont sans doute trop complexes dans le cadre de la première initiation et ce questionnement risque donc d'être peu productif.

En dehors de ce domaine, les types numériques sont assez peu adaptés pour le type de problème que nous envisageons. En effet, s'il est possible de codifier des données de nature diverse à l'aide de types numériques, le traitement a souvent, dans ce cas, recours à des fonctions mathématiques qui sont sans signification pour le problème posé. Dans l'exemple du paragraphe 3.3 , la représentation sous forme numérique du numéro INSEE conduirait à utiliser, pour isoler une partie du numéro, la division et le reste.

3.4.2 Les chaînes de caractères

Définition de la structure :

Nous avons indiqué ci-dessus que lors des premiers apprentissages, la

[63] Dans un langage de programmation, les nombres entiers ont nécessairement une valeur absolue limitée. Si un entier `A` positif est proche de cette limite, l'addition d'un autre entier `B` positif pourra soit entraîner une erreur, soit rendre un entier négatif. De même, la représentation des "nombres réels" en machine, sous forme d'une mantisse (nombre fractionnaire inférieur à 1) codée sous forme d'un nombre fixé de bits et d'un exposant variant dans un intervalle fini, permet de coder des nombres très petits et très grand, mais pour un nombre "grand" `A`, et un nombre "petit" non nul `x`, on peut très bien avoir `A + x = A`, ce qui est en contradiction avec l'arithmétique. Ce cas est étudié par exemple dans [MEYER BAUDOIN 80] (B.MEYER C.BAUDOIN. "Méthodes de programmation". Editions Eyrolles.) chapitre «introduction aux langages»

codification des éléments en jeu dans le problème ne peut se distinguer de la représentation dans le langage réellement utilisé, et que, en conséquence, il convient d'utiliser des représentations présentant un caractère suffisamment général ; c'est pourquoi nous limitons à 3 les fonctions sur les chaînes de caractères que nous utilisons avec les élèves (nous développerons ce point au chapitre 5, et nous présenterons la façon dont ces fonctions sont présentes dans les langages utilisés pour l'initiation):

- la fonction `sous-chaîne` a trois arguments : une chaîne, un ordinal (qui marque le rang du premier caractère de la chaîne résultat dans la chaîne argument), et un cardinal, représentant le nombre de caractères de la chaîne résultat.
- la fonction `longueur` dont l'argument est une chaîne, rend un nombre ; ce nombre peut être considéré comme un cardinal (nombre de caractères de la chaîne) ou comme un ordinal (rang du dernier caractère de la chaîne).
- l'opération de `concaténation` a deux arguments chaînes, et rend une chaîne .

Utilisation pour des problèmes :

Les chaînes de caractères ainsi définies permettent, dès l'initiation, de codifier des entités appartenant à des domaines sémantiques variés : des mots, des expressions , des listes linéaires d'objets représentés par des caractères ou des mots, et aussi des nombres considérés comme des chaînes de chiffres. La codification d'entités à l'aide de chaînes conduit à des traitements séquentiels typiques des traitements informatiques, et éloignés des traitements des mêmes entités par des opérateurs humains ; ainsi la recherche d'une occurrence d'une `sous-chaîne` dans une chaîne implique, dans le dispositif informatique, un traitement itératif et l'utilisation d'un index numérique, alors qu'un opérateur humain peut accéder directement au caractère cherché par appréhension globale de la chaîne.

Les traitements sur les chaînes utilisent la structure numérique sous les deux aspects, cardinaux et ordinaux : l'acquisition de la structure ordinale est intéressante à observer, en ce sens qu'elle est utile en informatique (calculs d'index sur les tableaux, par exemple), et qu'elle ne fait pas l'objet d'un enseignement au delà de la première année d'enseignement obligatoire (soit une dizaine d'année avant le début de l'enseignement d'informatique). Nous détaillerons ces propriétés de la structure de chaîne au chapitre 5. Dans la partie expérimentale de cette thèse, la structure de chaîne occupe une place importante puisque la partie II est une observation d'élèves en situation de résolution de problèmes impliquant la mise en oeuvre de cette structure, et la partie III un essai d'ingénierie didactique dans le même domaine.

3.4.3 Les booléens

Ils permettent de codifier des entités à deux valeurs, qu'il s'agisse d'objets ou de relations entre objets. Ils interviennent donc par exemple dans des problèmes concernant des dispositifs à deux états, et dans ce cas, la codification n'est pas immédiate pour le débutant, puisqu'elle est particulièrement réductrice. Organisés en tableau de booléens, ils interviennent dans des problèmes se ramenant à un parcours de graphe orienté (plan de ville, etc...); là aussi, la représentation spatiale familière, et le mode de raisonnement sur cette représentation s'éloignent de la représentation informatique et du traitement sur cette représentation, ce dernier traitement étant un calcul sur des booléens.

Par ailleurs, une conceptualisation adéquate des booléens paraît importante, puisque les traitements conditionnels (alternative, itération) font intervenir une condition qui est de type booléen. Jusqu'à ce que les élèves aient connaissance des booléens, le statut de cette condition, et des opérateurs propositionnels qui permettent de construire des conditions complexes reste flou, ce qui peut être gênant pour une utilisation systématique de ces structures (le passage d'une forme REPETER...JUSQU'A à la forme TANT QUE..., par exemple, conduit à considérer la négation d'une condition). Nous nous proposons de montrer dans la partie IV de cette thèse que des problèmes impliquant de représenter des objets du monde par des booléens contribuent à clarifier le statut des "conditions" auprès des élèves.

4. Conclusions du chapitre

Nous avons situé notre recherche au niveau des premiers apprentissages en informatique et indiqué que l'enjeu de ces apprentissages est l'accès des élèves aux concepts de base de l'informatique (variables, affectation, traitements alternatifs, traitements itératifs, typage des données...) leur permettant de dominer les premiers traitements dans des domaines de tâches et des langages variés, d'opérer les mises en relations, et les différenciations entre ces domaines et entre ces langages. Les concepts informatiques prenant leur sens d'abord dans des "problèmes de programmation" où ils interagissent, l'accès d'un débutant au niveau des concepts suppose qu'il se soit posé et qu'il ait résolu des problèmes divers, activité au cours de laquelle il se constitue un ensemble des représentations mentales que [HOC 77] désigne par Systèmes de Représentation et de Traitement (**S.R.T.**) . Pour nous, un **S.R.T.** est la structure mentale locale qui permet à un sujet, face à une tâche de programmation donnée, de construire une solution et/ou de vérifier sa validité. Le niveau des concepts est celui où le sujet, disposant de **S.R.T.** relatifs à des tâches diverses, est capable de les mettre en relation et de les différencier, de façon à opérer des transferts lorsque se présente une tâche nouvelle, et à pouvoir raisonner sur les concepts indépendamment des domaines de tâches.

Nous avons distingué des plan(s) relatif(s) aux objets.et des plan(s) relatif(s) aux traitements, s'organisant en sous-systèmes au sein d'un **S.R.T.**, de façon à permettre la prise en compte des spécificités des conceptions du sujet relatives aux objets, dans la résolution d'un problème donné à un moment donné de ses acquisitions, et des interactions entre les conceptions des objets et les conceptions des traitements. Dans le type d'initiation que nous envisageons, le mode le plus courant de constitution de **S.R.T.** à travers la résolution de problèmes, est l'importation de *plans* issus de **S.R.T.** adaptés à des traitements dans des domaines familiers (extra-informatiques). Il conduit à ce que les plans relatifs aux objets conservent dans le **S.R.T.** construit pour la recherche d'une solution informatique, des propriétés du contexte intuitif, et à ce que les plans relatifs aux traitements soient conçus de façon dépendante des objets sur lesquels ils portent.

Pour se constituer des **S.R.T.** adaptés, le sujet débutant devra donc être confronté à des problèmes de programmation qui, tout en permettant l'importation de plans à titre heuristique, lui imposent d'opérer des différenciations par rapport à ses conceptions des objets et traitements intuitifs. En particulier, il devra progressivement concevoir les objets informatiques comme *calculables*, et donc progresser vers l'utilisation de définitions abstraites, intégrer le caractère général des traitements...Nous en avons déduit que les énoncés doivent avoir les spécificités suivantes:

- le niveau où est posé le problème permet que les élèves considèrent de façon intuitive les objets en jeu et le lien entre données et résultats; plus précisément, le **S.R.T.** mis en oeuvre pour ce niveau permet à l'élève de se représenter l'état initial et l'état final, et contient des représentations schématiques de traitements.
- le niveau du traitement par le dispositif informatique impose, quant à lui, une "codification" de ces objets sous forme de données informatiques et la production d'une solution comme "calcul" sur ces données codifiées;
- les problèmes font interagir des concepts informatiques de base (objets, traitements) et au contraire, évitent de faire intervenir des éléments conceptuels (par exemple mathématiques) en dehors du champ de l'informatique.

Nous avons montré que les types numériques sont assez peu adaptés pour ce type de problème. Avec les chaînes de caractères et les booléens, il doit, au contraire, être possible de construire des problèmes utilisant ces types pour les codifications. Partant du cadre général tracé dans ce chapitre, le chapitre suivant précisera les hypothèses et le cadre spécifique pour l'étude expérimentale des parties II, III et IV.

Chapitre 4 Choix spécifiques pour l'expérimentation

hypothèses sur le fonctionnement cognitif et les problèmes
questionnement sur la situation pédagogique
l'option informatique comme terrain
planification de l'observation
méthodologie expérimentale

Au chapitre précédent nous avons indiqué dans quel cadre général s'insère cette thèse: choix du public observé, fonctionnement cognitif du sujet et problèmes de programmation en alphabétisation informatique. A l'intérieur de ce cadre général, le présent chapitre délimite le domaine d'investigation de notre thèse: hypothèses spécifiques sur le fonctionnement cognitif et les problèmes, questions concernant une situation pédagogique. La validation de ces hypothèses, et des réponses à ces questions supposent une étude expérimentale: nous décrivons donc également dans ce chapitre le cadre scolaire constituant le terrain pour cette étude expérimentale, nous discutons l'organisation dans le temps, et la méthodologie d'observation que nous adoptons pour cette étude expérimentale.

1. Hypothèses et questionnement

A la suite des conclusions du chapitre précédent, nous posons, pour le travail expérimental que constitue la suite de cette thèse, deux séries d'hypothèses et un questionnement:

1.1 Hypothèses sur les difficultés des élèves, et la constitution des S.R.T.

- **L'ampleur des difficultés** rencontrées chez une fraction notable des débutants (voir ci-dessous § 2.3) nous conduit à faire l'hypothèse que les concepts de base de l'informatique, contrairement à ce qui a été souvent considéré, ne sont pas disponibles de façon spontanée pour les débutants. Chez beaucoup d'élèves, la construction de représentations mentales adéquates du dispositif, s'organisant en **S.R.T.** différenciés et mis en relation, est un processus à la fois lent et nécessaire pour l'émergence des concepts. Ce processus implique un travail de mise en évidence et de remise en cause de **S.R.T.** non adaptés aux traitements informatiques.
- Dans les **S.R.T.**, nous distinguons les **plans relatifs aux objets** interagissant avec des **plans relatifs aux traitements**; les **plans relatifs aux objets** résultent de l'intériorisation par le sujet des propriétés des objets dans le domaine de tâche considéré, celles-ci étant influencées, à travers la résolution des problèmes, par les propriétés d'objets dans des contextes intuitifs (par exemple, la compréhension des propriétés des objets du type "chaîne de caractères" est influencée est influencée par la compréhension d'actions familières sur des textes): dans un **S.R.T.** construit pour une tâche de programmation dans un langage impératif (voir chapitre suivant), le plan relatif à un type d'objet est constitué des significations données par le sujet aux variables représentatives des objets de ce type, aux fonctions sur ces variables et aux éléments du langage liés à ces variables, en premier lieu **l'affectation**, et **les écritures fonctionnelles.** Les difficultés d'intégration de ces structures relatives aux objets, la spécificité des relations entre conceptions relatives aux objets et conceptions relatives aux traitements dans un **S.R.T.** donné, nous semblent avoir été négligés jusqu'ici (voir exemples chapitre 3 §2). Nous faisons donc l'hypothèse que la constitution de plans relatifs aux objets permettant la construction d'un traitement pour le dispositif est une étape obligée de la constitution de **S.R.T.** adaptés.
- L'interaction, dans la résolution d'un problème d'informatique, de la structuration des données et de la structure algorithmique, nous conduit à faire l'hypothèse que,

face à un problème mettant en jeu un traitement complexe (par exemple des alternatives imbriquées, ou une itération), **les plans relatifs aux traitements se construisent dans le S.R.T. du sujet en relation étroite avec les plans relatifs aux objets**, et donc que, tant que le sujet raisonne au niveau du domaine de tâches et non au niveau des concepts, **il ne peut envisager la création d'un traitement indépendamment de sa conception des objets informatiques en jeu**. Nous complétons donc l'hypothèse ci-dessus: la constitution de **S.R.T.** adaptés suppose de considérer et de faire évoluer de façon conjointe les plans relatifs aux objets et les plans relatifs aux traitements.

Conséquences de ces hypothèses:

Il nous faut vérifier en premier lieu que la codification d'objets intuitifs par des entités informatiques et les calculs simples (c'est-à-dire ne mettant pas en oeuvre des structures algorithmiques autres que la séquentialité) correspondant à des manipulations sur les objets intuitifs, ne vont pas de soi pour des débutants, qu'ils leur imposent un réel travail de construction de représentations mentales adéquates, un réel travail d'abstraction. Plus précisément, nous tenterons, au chapitre **5**, en ce qui concerne les chaînes de caractères, et au chapitre **10** en ce qui concerne les booléens, de cerner, à partir des éléments du langage, et de ce que nous pouvons prévoir du fonctionnement cognitif des élèves, des *difficultés*, traduction concrète de conceptions relatives aux objets inadéquates chez ces élèves. Puis, dans une étude expérimentale, nous examinerons comment ces *difficultés* peuvent ou non être repérées chez des élèves (chapitres **6** et **11**), et nous tenterons de mieux comprendre comment elles interviennent dans les conduites. Le terme de "*difficulté*" est volontairement vague. En effet, dans une phase d'alphabétisation qui s'étale sur peu de temps (rarement plus d'une année) et occupe relativement peu de séances, nous ne pouvons savoir a priori si ces conceptions erronées sont seulement transitoires chez le sujet, ou vont se transformer en *obstacles*. Nous pensons a priori, que les *difficultés*, en tant que constitutives d'un **S.R.T.** lié à un domaine de tâches donné, ont un ancrage local, et sont relativement résistantes, en ce sens qu'elles se retrouvent dans les conduites du sujet concerné, face à différents problèmes liés à ce domaine, et persistent malgré le "feed-back" de la machine ou de l'enseignant, tant que le sujet n'a pas reconsidéré la conception que la difficulté traduit. Nous soulignons que, constituant la première phase d'un enseignement d'informatique, l'alphabétisation ne saurait "hériter" de *conceptions* installées par des enseignements antérieurs, et qui feraient *obstacle* aux nouvelles connaissances, comme c'est souvent le cas en mathématiques. [64].

Puis nous devrons valider notre hypothèse concernant les conséquences pour la création des programmes par un sujet, de l'interaction, au sein de ses **S.R.T.** des plans relatifs aux objets et des plans relatifs aux traitements. Il nous faudra donc faire intervenir des problèmes impliquant des traitements plus élaborés que la simple séquentialité (itération, alternatives imbriquées), et observer comment les difficultés définies ci-dessus, vont se retrouver et interagir avec les difficultés propres au traitement complexe choisi, quelles nouvelles difficultés vont apparaître, liées aux particularités des représentations spontanées à partir d'une conception intuitive du traitement considéré.

1.2 Hypothèse sur les problèmes susceptibles de mettre à jour les difficultés et de faire évoluer les S.R.T.

Les éléments de représentation erronés dans les **S.R.T.**, que nous avons postulé ci-dessus, ont un domaine de validité non vide, et de nombreux énoncés peuvent en fait se situer dans ce domaine de validité, ou, plus simplement, ne pas les solliciter. Notre but est par conséquent de construire et de tester des énoncés où ces représentations peuvent être conçues par l'élève comme adéquates, mais se révèlent d'une manière ou d'une autre, entrer en contradiction avec la logique du dispositif informatique. Nous avons montré au chapitre précédent que ces énoncés ont les spécificités suivantes:

- le niveau où est posé le problème permet que les élèves considèrent de façon

[64] Pour une discussion sur les notions de conception et d'obstacle, voir [ARTIGUES 91]

intuitive les objets en jeu et le lien entre données et résultats;

- le niveau du traitement par le dispositif informatique impose, quant à lui, une **"codification" de ces objets sous forme de données informatiques et la production d'une solution comme "calcul" sur ces données codifiées;**
- les problèmes font interagir des concepts informatiques de base et au contraire, évitent de faire intervenir des éléments conceptuels (par exemple mathématiques) en dehors du champ de l'informatique.

L'hypothèse porte donc sur:

- la **possibilité de construire de tels problèmes** à l'aide des types de base des langages couramment utilisés pour l'initiation,
- l'utilité de ces problèmes pour la **mise en évidence du fonctionnement cognitif** des élèves,
- la possibilité de les organiser en "**progression**" visant à la mise en cause et à l'amélioration des **S.R.T.**

1.3 Questions posées par la situation pédagogique: le cas du travail dirigé en présence de l'ordinateur

La situation pédagogique, c'est-à-dire l'ensemble des interactions entre l'(les) élève(s), le maître, le problème de programmation, le dispositif informatique et les concepts, constitue un système complexe où le choix du problème (présenté ci-dessus) n'est qu'une variable parmi d'autres. En particulier, l'intervention du professeur ne saurait être évacuée de l'analyse, même si elle ne peut être caractérisée de façon immédiate dans la situation de "Travail Dirigé" qui domine l'alphabétisation: nous ne pensons pas que l'élève puisse seul, face au problème et au dispositif, construire de façon économique des représentations adaptées, et que le travail du professeur se réduise à une institutionnalisation des concepts a posteriori. L'absence de tradition en enseignement de l'informatique, particulièrement au niveau de l'alphabétisation et le nombre réduit d'études didactiques faisant de façon précise l'analyse des situations pédagogiques dans ce domaine, nous paraît rendre difficile la construction a priori de situations pédagogiques[65]. Plus que d'hypothèses, le travail sur ces situations nous paraît donc devoir, dans l'état actuel de la recherche, faire l'objet d'un questionnement. Les **situations de résolution par petits groupes, encadrées par l'enseignant, l'ordinateur étant à disposition pour des essais de compilation et/ou d'exécution** nous intéressent prioritairement. En effet, les pratiques d'alphabétisation informatique auxquelles nous avons participé avant cette thèse, nous conduisent à considérer ces situations comme les plus productives: les situations collectives, d'enseignement magistral, ou de résolution collective se heurtent en effet à l'hétérogénéité des représentations et des aptitudes qu'elles entraînent. Ceci constitue évidemment une impression subjective et non un résultat, mais, comme choix d'un *angle d'attaque*, la situation qui nous paraît la plus riche est aussi celle pour laquelle nous pouvons le plus facilement nous poser des questions:

- **la présence du dispositif informatique** est une spécificité de cette situation de résolution: comment les contraintes d'expression et les effets en retour par lesquels il se manifeste (résultats d'exécution, messages d'erreurs à la compilation et à l'exécution) interviennent-ils dans la construction des représentations des élèves?
- contrairement à d'autres situations de résolution, **l'enseignant** n'a pas un rôle d'organisateur de la communication. Pourtant il intervient auprès des groupes, sollicite, conseille... Dans les pratiques que nous connaissons, ces interventions sont

---

[65] L'existence d'un ensemble d'ouvrages de niveau universitaire présentant les savoirs en informatique nous a permis de repérer comme pertinente l'analyse de la forme sous laquelle le savoir est considéré (forme recontextualisée, forme décontextualisée) élaborée par [BROUSSEAU 86] pour la didactique des mathématiques. Le manque de référence que nous signalons rendrait hasardeux l'importation directe de la modélisation des situations pédagogiques qui est un autre aspect important du travail de G. BROUSSEAU. Cependant, à travers l'observation-questionnement que nous projetons, nous examinerons la pertinence, dans le domaine qui nous intéresse, de certains concepts crées pour cette modélisation (Chapitre 9)

très dépendantes des choix de résolution et de l'état d'avancement de chaque groupe, très liées également à l'interprétation qu'il fait des productions des élèves. Selon quels axes peut-on classer ces interventions ? Sur quels éléments l'enseignant peut-il s'appuyer pour interpréter les productions des élèves ?
- comment se fait la **communication** dans le groupe, et avec l'enseignant ?

2. Le terrain: l'option informatique des lycées; classe de Seconde et de Première

Nous avons expliqué au chapitre précédent pourquoi, dans l'état actuel de la didactique de l'informatique, la recherche doit partir de l'observation d'un public hétérogène sans finalité professionnelle. Dans cette perspective, l'option informatique des lycées (qui s'adresse à des élèves des sections classiques et de certaines sections techniques de l'enseignement long post-obligatoire) constitue un terrain adapté pour plusieurs raisons. D'une part, ayant dépassé le stade expérimental, les objectifs de cet enseignement ont été largement débattus et clarifiés, et font l'objet d'une publication officielle. D'autre part, la classe de Seconde est maintenant ouverte à la grande majorité d'une classe d'age, et bien qu'optionnel, l'enseignement s'adresse à tous: l'hétérogénéité qui doit en résulter est effective en classe de Seconde, sauf politique restrictive des établissements[66] . Les quelques éléments qui suivent visent à préciser certains aspects de l'option informatique de façon à délimiter la part de contingences qu'apporte le choix de ce terrain à l'étude expérimentale rapportée dans les chapitres suivants.

2.1 Programmes et pédagogies de l'option informatique

Les modalités d'organisation et les programmes de l'option sont des arrêtés ministériels publiés au bulletin officiel de l'Education Nationale et regroupés dans une brochure éditée par le CNDP [CNDP 87][67]. L'horaire est de 2h30, dont une heure de cours (plein effectif, soit en principe jusqu'à 40 élèves) et 1h30 de Travaux Pratiques (en demi-classe, avec présence des ordinateurs). Les objectifs généraux précisent qu'il s'agit d'un «*enseignement à caractère général*», donc non «*orienté vers un type d'application ou de profession*». Il s'organise en «*apport de connaissances techniques*», «*apprentissage de méthodes de travail*», et «*prise de conscience des enjeux économiques, sociaux et culturels*».

Le programme de la classe de Seconde recouvre les notions de base de l'informatique (données simples et leurs représentations internes et externes, traitements séquentiels, traitement conditionnel, traitement itératif). Les notions de tableau, de fichiers de données, de procédures et de fonctions sont au programme de Première.

Les directions pédagogiques officielles sont fortement influencées par le travail de J.ARSAC: méthode de résolution en 5 points («*Enoncé d'un problème; Découverte et expression d'un procédé de résolution; Codage dans un langage de programmation; Validation du programme ainsi obtenu; Utilisation effective pour l'application retenue*»), méthode d'enseignement faisant intervenir, comme point de départ, le développement d'«*exemples typiques*» par le professeur. Cependant, il nous semble que ces directions ne doivent pas être conçues comme un point d'achèvement normatif; le débat sur les méthodes et les contenus est largement ouvert, comme le montrent les articles parus dans la revue **Informatiques** (par exemple [PAIR 88], et les nombreuses contributions des professeurs de l'option).

---

[66] Devant l'afflux de candidatures pour l'option, certains établissement sont tentés d'opérer une sélection des élèves sur la base des acquis scolaires antérieurs, ce qui est en contradiction avec l'esprit et les textes de l'option. Bien qu'étant moins générale que ce que laisse supposer [M.E.N. 91], cette politique se rencontre. Nous nous sommes assurés que les classes où nous avons mené les observations n'ont pas fait l'objet d'une telle sélection.

[67] [CNDP 87] OPTION INFORMATIQUE Classes de seconde, première et terminale.CNDP 1987

2.3 Problèmes didactiques de l'option.

Nous avons dit que l'hétérogénéité est effective en classe de Seconde, sauf politique restrictive des établissement. Par contre selon [BARON 89][68], à partir de la classe de Première les élèves qui persistent dans cet enseignement sont en grande majorité des élèves de sexe masculin des classes scientifiques. Le taux important de non-continuation de l'option après la classe de Seconde, particulièrement marqué chez les élèves non-scientifiques et les filles, est évidemment un des problèmes importants, puisqu'il remet en cause l'option dans sa finalité d'enseignement ouvert à tous. Les professeurs de l'option sont conscients de ce problème: certains ([BENETOLLO 90][69]) insistent sur une prise en compte de l'hétérogénéité des élèves à partir de la situation pédagogique de Travaux Pratiques; d'autres ([AUTHIER, WAITER 89][70]) analysent les difficultés des élèves comme «liées au passage d'une situation concrète à sa représentation en termes d'objets abstraits». Ces analyses vont dans le sens des questions que nous nous posons au départ de cette thèse, mais, selon nous, la portée des débats est quelque peu limitée par le manque d'études réellement didactiques à support expérimental dans le domaine des premiers apprentissages en informatique.

2.4 Le cas des classes observées

Nous avons pu mener nos observation dans des classes où enseignaient des professeurs largement impliqués dans la réflexion pédagogique autour de l'option (un enseignant est le coordinateur académique, une autre encadre des formations pour les enseignants de l'option). Ceci explique que le programme et les choix pédagogiques adoptés dans ces classes sont en conformité avec les instructions officielles: les notions abordées en classe de Seconde, et l'hétérogénéité constatée se sont donc trouvées tout à fait adaptées à l'observation que nous projetions, au moins pour les parties concernant les chaînes de caractères (CF ci-dessous §1.1, 1.2 et 1.3). La répartition des notions entre la Seconde et la Première, ainsi que des contingences liées au langage utilisé dans les classes de Seconde auxquelles nous avions accès, nous ont conduit à mener l'observation liée au type booléen (CF ci-dessus §1.4) en classe de Première, au prix d'une moins grande hétérogénéité, dont il est important de tenir compte pour l'exploitation des résultats.

Comme toutes les classes de l'option, les classes observées utilisaient un langage "évolué" (symbolique). Nous avons vu au chapitre 3 (§ 3.1) que ceci n'est pas indifférent du point de vue de la constitution des **S.R.T.**: le sujet programmant en langage évolué ne peut en effet s'appuyer sur des représentations matérielles entièrement adéquates, ni sur des règles de fonctionnement totalement explicites, et doit donc utiliser les exemples proposés par l'enseignant, des connaissances issues d'apprentissages dans d'autres disciplines, les "effets en retour" du dispositif... Comme dans beaucoup de classes de l'option, le langage était impératif (BASIC ou PASCAL suivant les classes), l'important pour nous étant que le langage permette d'utiliser commodément les types de base (chaînes et booléens). Nous discuterons au chapitre suivant l'influence du choix d'un langage impératif, et la disponibilité des types de base dans les différents langages.

Nous n'avons pas particulièrement recherché des classes à faible effectif, mais il s'est trouvé que les classes où nous avons travaillé comptaient entre 13 et 15 élèves. Ce faible effectif s'est trouvé compatible avec les premières observations de type clinique (voir ci-dessous). Nous avons dû ensuite vérifier leur généralité par une épreuve portant sur plusieurs classes. La classe où nous avons expérimenté une progression d'exercices (Cf 1.3 Essai d'"ingénierie didactique") disposait de la

---

[68][BARON 89] G.L. BARON L'option informatique à la rentrée de 1988/89 in INFORMATIQUES n°6 2nd trimestre 89.

[69][BENETOLLO 90] R. BENETOLLO Hétérogénéité et réussite dans les classes d'option informatique des lycées Actes du 2ème colloque francophone sur la didactique de l'informatique. Septembre 90.

[70][AUTHIER, WAITER 89] A. AUTHIER, N. WAITER Modèles, objets et pédagogies de l'option informatique in INFORMATIQUES n°6 2nd trimestre 89.

structure "Travaux Pratiques" (ordinateur pour chaque groupe de 2 élèves) sur la totalité des 2h30 hebdomadaires, ce qui a constitué un élément facilitant pour cette expérimentation (l'heure hebdomadaire de cours n'aurait pas eu d'intérêt dans cette expérimentation compte tenu des choix que nous avons indiqué en 1.3).

3. Planification de la recherche

Les hypothèses à tester, et le questionnement concernant la situation pédagogique de travail dirigé (§1) conduisent à deux types types de travaux énoncés ci-dessous en 3.1 et 3.2.

3.1 Observer de façon ponctuelle à des moments opportuns, des élèves ayant constitué, dans un cadre pédagogique donné, des ébauches de représentations

Cette observation a pour but de vérifier la pertinence des hypothèses concernant le fonctionnement cognitif et les *difficultés* qui en résultent (ci-dessus 1.1). Pour cette observation, le *type de problème* que nous avons défini en 1.2 est évidemment mis à contribution. La discussion du chapitre précédent concernant l'utilisation des types prédéfinis, et l'ordre dans lequel ces types sont abordés dans les classes conduit à centrer une première observation sur les problèmes utilisant les chaînes de caractères et les types associés (ordinaux et cardinaux), puis à l'élargir aux booléens. Cette observation comporte donc trois temps:

3.1.1 Observation de "*difficultés*" liées aux structures relative aux objets dans le cas de tâches de programmation sur les chaînes de caractères

Nous devons faire des hypothèses sur les *difficultés* spécifiques, choisir une classe et le moment adapté (il faudra que les élèves aient commencé des exercices sur les chaînes, mais pas nécessairement les traitements complexes), prévoir une épreuve et des entretiens à partir d'une série limitée d'exercices (cette observation est rapportée en partie **II** chapitre **6**). Les exercices, conformes aux spécifications énoncées en 1.2 devront faire intervenir la structure de chaîne, l'affectation et la séquentialité, que nous pensons constitutifs des plans relatifs aux objets, et exclure les "traitements complexes" (itération, alternatives), de façon à séparer les *difficultés* observées de celle qui interviennent dans le cas d'un traitement complexe..

3.1.2 Observation de l'interaction dans un S.R.T., des plans relatifs aux objets, et des plans relatifs aux traitements dans le cas d'un traitement sur une chaîne de caractères

Il nous faudra ici choisir un type de traitement algorithmique interagissant avec la structure de chaîne, faire des hypothèses sur la façon dont le fonctionnement cognitif des élèves face à ce type de traitements peut se traduire en conduites, construire une épreuve et faire passer des entretiens. (observation rapportée en partie **II** chapitre **7**)

3.1.4 Par la suite, il nous faudra aussi montrer que les résultats observés dans ce cadre ne sont pas limités aux chaînes de caractères

Au chapitre précédent, nous avons examiné les possibilités d'exploitation des codifications utilisant les booléens: codification d'objets à deux états à l'aide de variables booléennes, codification de relations à l'intérieur d'ensembles d'objets à l'aide de tableaux de booléens. Il parait donc judicieux de prolonger le travail sur les chaînes par une observation d'élèves en situation de résolution de problèmes impliquant l'emploi de variables booléennes, les objectifs de cette observation étant, comme pour les chaînes, de mettre en évidence des plans relatifs aux objets booléens, les difficultés crées par l'existence de conceptions erronées, leurs spécificités, leur interaction avec les plans relatifs aux traitements, leur apport aux apprentissages généraux en informatique...

3.2 Tester la possibilité d'organiser la *classe de problèmes* que nous avons définie ci-dessus, en *progression* visant à faire évoluer les S.R.T. des élèves, et voir comment répondre aux questions concernant la situation pédagogique

Ceci conduit à

3.2.1 un travail d'"ingénierie didactique"

Après les observations envisagées en 3.1.1 et 3.1.2, et en fonction de la compréhension du fonctionnement cognitif des élèves qu'elles nous auront apporté, il nous sera possible, en faisant le choix d'un champ de connaissances bien délimité, de construire un ensemble de problèmes de programmation, organisé en **progression** et visant à des acquisitions dans ce champ. Ce travail de construction nous conduira à analyser de façon plus précise que dans ce chapitre, les choix possibles concernant les énoncés, et à classer ces énoncés suivant une typologie relative aux aspects du fonctionnement cognitif des élèves sur lesquels il nous semble possible d'intervenir. (partie **III** chapitre **9**)

3.2.2 une observation d'élèves à partir de la progression ainsi construite

Cette observation doit nous permettre d'une part d'examiner les points forts et les points faibles de la progression et donc de l'analyse qui y a conduit, et d'autre part, de permettre un examen des questions posées en 1.3 concernant la situation pédagogique de travail dirigé par petits groupes; il nous faudra préciser le choix des élèves observés, le choix du moment dans les apprentissages où s'insère cette progressions, le choix la procédure d'observation. (partie **III** chapitre **10**).

4. Méthodologie

4.1 Les choix méthodologiques possibles

Une recherche en didactique suppose en premier lieu que les hypothèses avancées, les questions posées, le soient à partir d'une étude du domaine scientifique concerné, et des conditions psychologiques de l'apprentissage des notions de ce domaine. C'est ce que nous avons tenté de faire jusqu'à ce point de notre thèse. La recherche suppose également que ces hypothèses et questions soient confrontées à la réalité, sous la forme d'une étude expérimentale: seule cette confrontation permet de décider de la validité de ces hypothèses, de savoir si les questions ont été bien posées, quelles hypothèses supplémentaires on peut en attendre. Le cadre même des questions que nous nous posons conduit de façon directe au choix du terrain expérimental: nous avons montré au chapitre précédent que le terrain des premiers apprentissages est, dans l'état actuel de la recherche le seul possible, et dans ce chapitre, que l'option informatique des lycées convient tout à fait. Il nous reste à préciser le type d'étude expérimentale qui convient à nos hypothèses et questions: schématiquement le chercheur peut mener deux types d'études:

- une étude clinique: celle-ci consiste à repérer des conduites, à les classer, mais également à approfondir par une étude spécifique auprès d'élèves choisis, les raisons des ces conduites, de façon à les relier aux hypothèses concernant le fonctionnement cognitif.
- une étude quantitative: elle consiste à recueillir des données sur un nombre important d'élèves, concernant des conduites reliées aux hypothèses cognitives , de façon à en montrer la généricité, à établir des relations entre facteurs.

L'étude clinique convient particulièrement lorsque la définition des hypothèses, le lien entre les aspects du fonctionnement cognitif sur lesquels elle porte et les conduites, sont problématiques. Il faut souligner que dans ce cas, le processus d'élaboration des hypothèses et la confrontation à la réalité se conduisent dialectiquement: les premières conduites observées conduisent à approfondir le cadre d'analyse des processus cognitifs et donc à préciser les hypothèses, ce qui, à son tour, permet de mieux centrer l'observation. L'étude clinique, ne prétendant pas à la vérité statistique, peut se mener sur relativement peu de sujets, à condition de repérer les particularités

(apprentissages antérieurs...) de ces sujets, et d'éviter des cas trop particuliers pouvant interférer entre le fonctionnement cognitif et les conduites. Dans le cas où la problématique est clairement établie, et les conduites clairement reliées aux processus cognitifs, l'étude quantitative permet d'établir des vérités de nature statistique.

Le type d'activité des élèves, support de l'observation, compatible avec le cadre cognitif dans lequel nous nous situons est la résolution de problèmes de programmation. Ceci suppose donc que l'observation qu'elle soit de nature clinique ou quantitative, parte d'énoncés proposés aux élèves. (nous désignerons dans la suite de cette thèse, par *épreuve* , un ensemble d'énoncés proposés aux élèves). Nous verrons que, selon les contenus visés, la totalité de la tâche d'élaboration d'une solution peut être à la charge de l'élève, ou, au contraire, certains éléments de résolution (en particulier concernant le codage de l'information) sont précisés dans l'énoncé, par exemple sous la forme d'une amorce de programme, et la tâche de l'élève est de compléter ce programme à partir de sa compréhension des éléments déjà présents. Il est clair qu'une étude de type quantitatif ne pourra laisser trop d'éléments ouverts si l'on veut pouvoir procéder à un dépouillement statistique. D'autre part, une étude de type quantitatif peut difficilement prendre en compte des réponses d'élèves résultant d'une activité de programmation allant jusqu'aux essais de programmation sur le dispositif: ainsi, elle peut seulement apporter des informations sur une conception à un moment donné, sans que le "feed-back" apporté par la confrontation au dispositif puisse être pris en compte.

Au contraire, une épreuve visant à une étude clinique pourra être plus ouverte, le dépouillement pouvant prendre en compte de façon descriptive les différentes démarches. D'autre part, dans le cas d'une étude clinique, l'épreuve sur papier n'est qu'un point de départ: elle peut être complétée par un "*entretien*" qui consiste, à l'issue de l'épreuve , à demander aux élèves d'entrer en machine la solution qu'ils ont produite par écrit, puis de poursuivre leur recherche sous forme de travail dirigé. Ceci permet, à partir d'enregistrement au magnétophone des interventions des élèves, et des indications orales qui leur sont données, ainsi que de la trace écrite des différents essais de programme, erreurs et résultats d'exécution, de mieux prendre en compte la démarche de résolution des élèves, et ses évolutions éventuelles.

4.2 La méthodologie adoptée pour chacun des axes de l'étude expérimentale

Dans le cas de notre thèse, il est clair que le cadre d'analyse était à construire. Les premières observations ont donc été contemporaines de l'élaboration des premières hypothèses. Par la suite, le cadre d'analyse s'est affirmé, mais nous avons voulu continuer à prendre en compte les démarches de résolution des élèves, et amorcer une étude de la situation pédagogique. Ceci explique qu'une grande part de l'étude expérimentale de cette thèse soit de type clinique, à une exception près.

4.2.1 Premières observations cliniques sur les chaînes de caractères

Les premières observations (objectifs énoncés en 3.1.1) menées au cours de l'année scolaire 86-87 ont été de nature clinique. Nous n'avons pu, en effet, élaborer que très progressivement un cadre d'analyse pour l'interprétation des difficultés rencontrées par les élèves au cours des séances de travaux dirigés par le professeur et dans l'épreuve sur papier (EPR1), que nous leur avons fait fait passer au cours du mois de janvier (voir chapitre 6). Le petit nombre d'élèves dans la classe (11 à 13) nous a permis de classer finement ces difficultés, de les mettre en relation avec l'enseignement reçu et de repérer certains axes. Nous avons vraiment commencé à les situer après le dépouillement de séries d'*entretiens* que nous avons fait passer à trois groupes d'élèves.

Par la suite, dans la même classe, nous avons mené l'observation EPR2 (voir chapitre 7) de la même façon que EPR1. Le cadre d'analyse que nous avions commencé à élaborer s'est révélé productif, car nous avons pu cette fois interpréter les résultats obtenus dès le dépouillement de l'épreuve sur papier, la série d'entretiens que nous avons fait passer à un groupe d'élèves venant seulement en confirmation.

4.2.2 Une étude quantitative sur les chaînes de caractères

Nous avons voulu ensuite vérifier auprès d'un nombre plus important d'élèves le caractère de généralité des *difficultés* rencontrées. C'est pourquoi, en fin d'année 87-88, nous avons fait passer dans trois classes de deux lycées différents une épreuve (EPR3) reprenant des items analogues à EPR1 et EPR2. L'évaporation des effectifs dans ces classes en fin d'année nous a permis seulement de recueillir une cinquantaine de réponses, la population concernée offrant cependant une diversité suffisante en matière d'enseignement reçu, de conditions pédagogiques, et d'aptitudes scolaires pour que les résultats soient significatifs. Nous avons pu en effet confronter les difficultés rencontrées à l'enseignement reçu et aux aptitudes générales. (Voir Chapitre **8**).

4.2.3 Le travail sur les booléens

Nous avons mené le travail sur les booléens (Objectifs énoncés ci-dessus en 3.1.3) de la même manière que les premières observations sur les chaînes (épreuve sur papier suivie d'entretiens). Cette méthode clinique s'est en effet révélée bien adaptée à la construction d'un cadre d'analyse des conduites des élèves, à partir d'hypothèses cognitives. L'objectif n'est pas en effet, l'exhaustivité, mais la mise en évidence de ce que le cadre d'analyse que nous avons mis en place pour les chaînes peut se généraliser à d'autres types.

4.2.4 Choix pour l'ingénierie (travail sur les problèmes à partir des difficultés repérées, tentative de caractérisation d'une situation pédagogique)

Le travail d'ingénierie didactique dont les objectifs ont été énoncés en 1.3, et que nous présentons au chapitre **9** est conçu comme une progression d'exercices devant faire travailler les élèves sur un dizaine de séances, d'une durée de une heure à une heure et demi sous forme de Travail Dirigé. Portant sur le structure de chaîne, et ne visant donc pas l'ensemble des objectifs de la classe de Seconde, cette progression trouve sa place à un moment où les élèves ont commencé à aborder la structure itérative. Nous avons fait une pré-expérimentation au cours de l'année scolaire 87-88, puis une observation précise au cours de l'année scolaire 88-89; voulant comprendre en profondeur les conduites et l'évolution d'élèves face à ces énoncés, et analyser la situation pédagogique (résolution par petit groupe en présence de l'ordinateur avec intervention d'un enseignant), nous avons, ici aussi, fait le choix d'une étude clinique. Ce type d'étude implique que les séances de résolution fassent l'objet d'un enregistrement précis (magnétophone + trace des écrans), que les interventions pédagogiques soient maîtrisées, et que l'environnement de travail habituel ne soit pas perturbé. Dans notre cas, travaillant seul à l'observation et au dépouillement, il n'était pas envisageable de suivre plus de deux groupes de deux élèves. Nous avons donc demandé au professeur de pouvoir travailler avec deux groupes constitués d'élèves particulièrement en difficulté dans les apprentissages antérieurs. L'étude réalisée (chapitre **10**) est marquée par les difficultés importantes (bien qu'assez diverses) que rencontrent ces quatre élèves. Elle constitue un premier repérage de l'évolution des difficultés, et des caractéristiques de la situation pédagogique.

Chapitre 5 Le débutant et le langage de programmation

les structures générales d'un langage impératif
le type de données "chaînes de caractères"

Notre hypothèse est que la constitution de représentations mentales adéquates du dispositif informatique peut-être, chez le débutant, un processus à la fois lent et indispensable pour l'émergence des concepts; nous avons dit également qu'une bonne connaissance des conditions dans lesquelles ces représentations se constituent, doit permettre de construire des progressions pédagogiques où les élèves rencontrent des difficultés liées à la distance entre une conception spontanée (naïve) du traitement des données et le traitement par un dispositif informatique, et où ils les résolvent progressivement. Nous précisons dans ce chapitre les caractéristiques de base des langages et comment nous voyons leur utilisation par les élèves, de façon à disposer d'un cadre d'analyse pour, dans un premier temps, préparer les problèmes sur lesquels nous projetons de faire travailler les élèves, et, après ce travail, classifier les difficultés qu'ils auront effectivement rencontrées dans ces problèmes.

Les problèmes où les données et résultats sont déjà *codifiés*, c'est-à-dire de façon générale, les problèmes portant sur l'emploi de types numériques ou dérivés et où ces types ne se distinguent pas des entités mathématiques qu'ils codifient ne nous semblent pas les mieux adaptés pour la mise en évidence de la distance entre le traitement habituel (c'est-à-dire mathématique, dans ces problèmes), et le traitement informatique. Une de nos hypothèses est donc que des problèmes portant sur d'autres types de données permettent mieux la prise en compte de cette distance. C'est pourquoi nous avons choisi dans un premier temps d'étudier la résolution par les élèves de problèmes impliquant l'emploi des chaînes de caractères. Dans ce chapitre, nous étudions donc d'une part les caractéristiques générales des langages susceptibles d'être utilisés par les élèves, d'autre part la structure de données "chaînes de caractères", puis nous examinons comment ces caractéristiques sont présentes dans les langages de programmation réellement existants. Ce chapitre est accompagné d'une annexe (annexe **5**) qui détaille certaines des références utilisées dans le chapitre : il s'agit d'une part de la présentation de certains textes concernant l'itération (discussion à partir des ouvrages d'enseignement de la programmation de J.ARSAC, présentation de travaux didactiques), d'autre part d'une présentation détaillée des caractéristiques de différents langages (BASIC, LSE, PASCAL, LOGO,...) et d'une note historique sur l'utilisation des chaînes de caractères en informatique.

1. Les structures fondamentales des langages utilisés lors des observations. Affectation, notation fonctionnelle, itération à un point de sortie.

Parmi les structures fondamentales, nous étudions d'une part les actions élémentaires et la notation fonctionnelle ; pour nous, ces éléments interviennent dans les représentations du sujet relatives aux objets. Nous étudions d'autre part les "actions complexes", particulièrement l'itération à un point de sortie, qui, pour nous, interviennent dans les représentations du sujet relatives aux traitements.

1.1 Les actions dans les langages impératifs; la notion d'assertion

Dans les observations que nous rapportons aux chapitres suivants, les élèves utilisent des langages impératifs, BASIC pour certaines classes, PASCAL pour d'autres; ils utilisent seulement les structures élémentaires de ces langages, qui diffèrent peu d'un langage à l'autre. C'est pourquoi le même type d'observation peut être mené avec une variété de langages dans la classe des langages impératifs accessibles aux débutants, classe qui comprend également LSE, des langages algorithmiques et les langages de commande des outils généraux (S.G.B.D.,

tableurs...)71. Nous étudions donc ici les structures fondamentales, communes à cette classe de langages.

La classe des langages impératifs se caractérise par la rédaction du programme comme suite d'actions, et de définitions d'actions, à la différence des langages fonctionnels où le programme se présente comme la définition et la composition de fonctions. Un programme dans un langage impératif étant une suite d'actions, les états du dispositif en divers points du programme se décrivent comme une succession d'assertions, c'est à dire de relations logiques portant sur les variables du programme. Considérons par exemple ci-contre le programme d'exponentiation, (x étant un nombre quelconque, et a un entier positif , calculer le produit xa):

```
n← 0
r← 1
FAIRE
        SI n=a EXIT
        r← r * x
        n← n + 1
BOUCLER
```

Nous donnons un sens aux actions dans le programme en plaçant des assertions (entre crochets, en italiques):

```
[[ VRAI]]
n← 0
[[n = 0]]
r← 1
[[n = 0  r = 1]]
FAIRE
  SI n=a EXIT
  [[n < a   r = xn]]
  r← r * x
  [[n < a   r =xn+1]]
  n← n + 1
  [[n ≤ a   r = xn]]
BOUCLER
[[n = a   r = xn]]
```

Au départ, on a l'assertion la plus générale ([[VRAI]]); chaque action fait passer d'une assertion à l'autre selon une loi qui tient à sa nature (nous préciserons ces lois plus loin dans le chapitre). Une action du programme étant donnée, l'assertion qui la précède est appelée pré-assertion, l'assertion qui la suit est appelée post-assertion. Le choix des assertions n'est pas unique; il dépend de la signification que l'on veut donner au programme. Dans l'exemple ci-dessus, on jugera que dans l'assertion finale la présence de n n'est pas fondamentale, et on remplacera cette assertion par l'assertion plus faible: `[[ r=xa ]]` . De façon générale, si `[[P]]A[[Q]]` et si `P'` ⇒ `P` et `Q` ⇒ `Q'` alors `[[P']]A[[Q']]`.

Cette caractérisation de l'action comme transformateur d'assertions permet de considérer le programme comme une suite de transformations des relations logiques entre les variables, et pas seulement comme suite de changements de valeur de ces variables; on remarquera en particulier que les assertions sont établies en des points du programme, et non en des moments de l'exécution: dans une instruction exécutée plusieurs fois, l'assertion doit être valide pour toute exécution. Cette définition a été introduite pour les preuves de programmes, et se révèle extrêmement féconde pour l'interprétation des programmes en programmation impérative. On pourra en trouver une présentation complète dans [ARSAC 83] ou dans [DJISKRA 76]. Nous l'utilisons pour caractériser la signification des actions élémentaires des langages utilisés par les élèves; en effet, comme nous le verrons, l'interprétation naïve de ces actions est porteuse de nombreux contresens.

---

71 Les langages impératifs se caractérisent par l'emploi de l'affectation pour l'évolution des relations entre variables, et de l'itération comme structure algorithmique. Nous avons indiqué au chapitre précédant (§2.4) que l'important pour nous est que le langage permette de manipuler commodément les types de base (chaînes et booléens): nous discuterons par conséquent à titre d'exemple de langage non-impératif le traitement des données "textuelles" et booléennes en LOGO.
Le cadre d'hypothèses et la méthodologie d'observation définies dans les deux chapitres précédents, sont conçus pour ne pas dépendre d'un langage. Nous nous attendons à trouver de façon expérimentale des conduites d'élèves conformes à nos hypothèses, ces conduites pouvant cependant différer en fonction des particularités de chaque langage (nous verrons en particulier l'influence de la forme structurée de l'itération).
Le S.G.B.D. (Système de Gestion de Base de Données) **Dbase (Aston Tate)** comprend un langage de commande très proche des langages impératifs que nous considérons, mais avec des objets spécifiques (fichiers, enregistrements, index...). L'organisation d'un tableur, comme outil de programmation, bien que sa structure puisse être considérée comme impérative, diffère notablement d'un langage de programmation.

Dans un langage impératif "pur", plutôt que des fonctions, on trouverait des actions ayant un (des) paramètre(s) donnés, et un (des) paramètre(s) résultats Par exemple, l'action additionner aurait deux paramètres donnés et un paramètre résultat (la somme des deux données); l'action diviser aurait deux paramètres données, et deux paramètres résultats (le quotient et le reste). Il n'y a pas d'"opérations" numériques, car ces opérations sont en fait des fonctions à deux variables. L'affectation est une action ayant un paramètre donné, et un paramètre résultat. Les entrées et l'affectation d'une constante sont des actions ayant seulement un paramètre résultat. Les sorties ont seulement un (des) paramètre(s) données; ne modifiant pas l'état des variables, elles n'ont pas d'influence sur les relations logiques entre variables du programme. Les programmes rédigés dans un langage impératif "pur" seraient lourds, du fait de l'impossibilité d'utiliser les raccourcis que permet la composition des fonctions. C'est pourquoi les langages que nous considérons ne sont pas des langages impératifs "purs": ils disposent en pratique seulement des actions d'affectation, d'entrée et de sortie. Les autres actions sont réalisées par l'affectation et un (des) appel(s) de fonction; ainsi, une action telle que `somme(a,b,c)`, dont `a` serait le paramètre résultat, et `b` et `c` les paramètres données se réalise par l'affectation à `a` de la somme `b+c`, que l'on note `a←b+c`. Le programmeur dispose également de la possibilité de définir des actions et de leur associer un identificateur, en BASIC par exemple, il s'agit de sous-programmes désignés par un numéro de ligne, en PASCAL et LSE, de procédures. Mais cette possibilité n'est pas abordée dans le type d'initiation que nous considérons ici. Egalement, le programmeur dispose de la possibilité de créer des actions complexes (alternatives et itérations) à l'aide d'avertisseurs adéquats; ces actions complexes permettent la construction d'algorithmes non-triviaux, et c'est évidemment à partir de ces actions complexes que les notions informatiques prennent leur sens. En particulier, les 4 articles que nous avons étudié au chapitre 1 analysent l'activité de programmation dans le cadre de ces actions complexes: T.H. GREEN étudie la compréhension des structures alternatives, R. SAMURCAY et E. COHORS situent leurs observations dans le domaine de l'itération, et J.M. HOC étudie l'activité de programmeurs pour un problème impliquant la coordination de diverses formes d'actions complexes. Ces études, et d'autres, nous conduisent à penser que, pour des débutants, la construction d'algorithmes non-triviaux par la mise en oeuvre d'actions complexes ne peut s'acquérir que très progressivement, c'est pourquoi notre étude portera seulement sur une catégorie limitée d'itération (l'itération à un point de sortie).

Dans les langages classés "impératifs" que nous considérons ici, le programmeur débutant dispose donc essentiellement des actions d'affectation et d'entrée/sortie, et utilise les fonctions prédéfinies. Il peut, de plus, former des actions complexes, soit alternatives, soit itératives. Dans les paragraphes qui suivent, les spécificités de chacun des ces éléments du langage sont analysées, en lien avec les acquis antérieurs des élèves.

1.2 Les actions simples: l'affectation et les entrées/sorties

Nous avons vu ci-dessus que l'instruction de **sortie** ne modifie pas les relations logiques entre les variables du programme. Une instruction d'**entrée** située dans le corps du programme "libère" la variable sur laquelle elle porte, en ce sens qu'elle détruit les relations logiques qui pouvaient exister entre cette variable et les autres variables du programme et n'en introduit pas de nouvelles. En fait, dans beaucoup de problèmes, les instructions d'entrée peuvent être placées en début de programme; en ce point, il n'existe pas de relations logiques entre les variables, et l'instruction d'entrée n'en définit pas. Les problèmes où des entrées doivent nécessairement se situer dans le corps du programme, sont ceux où il s'agit de traiter des entrées de données répétitives, par exemple, la lecture et le traitement de données à partir d'un fichier séquentiel. Nous nous limiterons, sauf dans deux cas particuliers (chapitre 9) aux problèmes où les entrées de données peuvent être situées en début de programme.

1.2.1 La sémantique de l'affectation

L'affectation est dans le premiers exercices que nous proposerons aux élèves la seule action ayant une influence sur les relations logiques entre variables du

programme. E. DJISKRA ([DJISKRA 76]) définit l'affectation x ← E où E est une expression (contenant éventuellement des variables), , comme l'action telle que l'assertion la plus générale qui est transformée en l'assertion Q est obtenue en remplaçant toutes les occurrences de x dans Q par l'expression E. Plus précisément, si l'on note $Q_{E->x}$ ce prédicat, on a:

$[[Q_{E->x}]]$ x← E $[[Q]]$

et $[[P]]$ x← E $[[Q]]$ si et seulement si $P \Rightarrow Q_{E->x}$

E. DJISKRA montre que cette définition correspond bien à l'idée intuitive de l'affectation. Ainsi: pour l'affectation d'une constante, par exemple x←7 , l'assertion la plus générale (la plus faible) transformée en $[[x=7]]$ est l'assertion toujours vraie $[[7=7]]$, donc, partant de tout état initial, l'assertion $[[x=7]]$ est établi après l'affectation et il n'existe aucun état initial conduisant à $[[x=8]]$; pour l'affectation d'une expression dépendant d'une autre variable, par exemple x← 2*y + 1 l'assertion la plus générale conduisant par exemple à $[[x=3]]$ est $[[2*y+1=3]]$ qui est équivalent à $[[y=1]]$; de même, pour l'affectation x←x * x, l'assertion la plus générale pour que $[[x=4]]$ est $[[x = 2 \text{ ou } x = -2]]$..

1.2.2 L'affectation pour l'élève débutant

Si l'affectation a un effet fondamental en modifiant les relations entre les variables du programme, c'est-à-dire l'état interne du dispositif, pour les élèves, elle ne produit rien de visible extérieurement; par contre, les actions de sortie constituent des effets visibles extérieurement, alors qu'elles sont indifférentes du point de vue des relations logiques entre variables du programme. Les élèves vont ils comprendre le rôle de l'affectation dans l'évolution de l'état du dispositif ? Ou vont-ils penser que d'autres éléments du langage tiennent ce rôle ?

Une présentation a-priori de l'affectation comme transformateur de prédicats (c'est-à-dire comme marquant l'évolution des relations logiques entre les variables) serait exacte du point de vue scientifique, mais visiblement inabordable par des débutants. On présente donc généralement l'affectation en faisant appel aux connaissances des élèves sur les variables en mathématiques et/ou en s'appuyant sur les représentations des élèves sur un dispositif de bas niveau, où la mémoire serait un ensemble de cases dont le contenu peut être remplacé (on dit que la mémoire est un ensemble de "tiroirs"),

1.2.3 Les précurseurs de l'affectation en mathématiques

Les variables en mathématiques sont plutôt des inconnues que des quantités qui varient. Il a été noté à juste titre ([PAIR 88]) que les connaissances sur les variables en mathématiques sont des précurseurs facilitants dans certains cas, inhibants dans d'autres. Il nous semble que l'affectation c← a + 2 , par exemple, peut s'apparenter à la désignation en mathématiques d'une expression comportant une inconnue par une nouvelle lettre. On attendra des élèves par exemple qu'ils posent c=a+2 si on leur demande de chercher le centre d'un cercle d'équation $x^2 - 2(a+2)\ x + y^2$. On peut également chercher des précurseurs à des affectations telles que x←2*x du côté des changements d'indice dans les sommations, ou du côté des changements de variables d'intégration (la somme des nombres pairs inférieurs à N, est égale à 2 fois la somme des entiers inférieurs à N/2), mais ces précurseurs ne concernent pas les élèves que nous observons ici. La spécificité la plus notable de l'affectation en informatique est la prise en compte de temporalité d'exécution : par exemple, on distinguera la variable c ayant reçu par affectation la valeur a + 2 de la fonction c qui rend a + 2 : en effet, la valeur de la variable c n'est pas modifiée par un changement de la valeur de a, et la variable c dépend donc de la valeur de a **au moment** de l'affectation, alors que la valeur **courante** de a doit être prise en compte à chaque appel de la fonction c.

1.2.4 La représentation de la mémoire comme ensemble de tiroirs

La représentation de l'affectation à partir des cases mémoires montre l'évolution de chaque variable et permet ainsi de simuler le déroulement pas à pas d'un programme. Elle nous semble insuffisante, car trop locale: en effet, elle ne met pas en évidence

l'évolution des relations logiques entre variables. Par exemple, l'affectation x←x+1 modifie x, et la variable x peut être facilement comprise comme un compteur "s'incrémentant". Mais cette affectation modifie également les liens logiques entre variables: si par exemple, il était établi avant l'affectation que *[[y=x+3]]*, alors, après l'affectation, *[[y=x+3]]* est faux, et il est par contre établi que *[[y=x+2]]*. De même, si avant l'affectation x← 3, on avait une relation logique entre deux variables x et y, par exemple *[[x = y * y]]*, l'affectation considérée détruit cette relation.

Les précurseurs en mathématiques pourraient donc donner une idée plus juste de l'affectation, mais l'article de R. SAMURCAY (article étudié en chapitre 1) nous semble montrer que, pour des élèves comparables à ceux que nous projetons d'observer, ces précurseurs sont insuffisants pour la construction de programmes utilisant l'itération; en effet, les élèves éprouvent des difficultés à passer d'un schéma de calcul connu en mathématiques à un schéma de calcul algorithmique (l'absence de temporalité d'exécution en mathématiques pouvant être une explication de cette difficulté).

1.3 La notation fonctionnelle

Nous avons noté que, l'affectation étant la seule véritable action, pour obtenir de nouvelles relations logiques entre variables, les langages comportent un ensemble de fonctions qui peuvent être combinées avec les identificateurs de variables et les constantes pour former l'expression de la partie droite de l'affectation. Cet ensemble de fonctions se subdivise en plusieurs jeux, chacun de ces jeux étant relatif à un type de donnée. Nous étudierons plus particulièrement au sous-chapitre suivant le jeu de fonctions relatif au type "chaînes de caractères". Ces fonctions ont un ou plusieurs arguments, et peuvent être composées: par exemple, en BASIC (voir en annexe 5 §4.1 une présentation de ce langage), pour affecter à une variable A$ le caractère du milieu d'une chaîne C$ de longueur impaire, on écrira:

```
LET A$ = MID$(C$,(LEN(C$)+1)/2,1)
```

La fonction MID$ (sous-chaîne) a trois arguments; elle est donc composée au niveau de son deuxième argument, avec la fonction 'diviser', elle même composée au niveau de son premier argument avec la fonction 'additionner', elle même composée au niveau de son premier argument avec la fonction LEN (longueur). On a donc, dans cette expression une composition sur quatre niveaux. L'affectation à une (des) variable(s) intermédiaires permet de séparer ces niveaux:[72]

```
LET L= LENGTH(C$)
LET PosiMilieu= (L+1)/2
LET A$= MID$(C$, PosiMilieu, 1)
```

On remarquera sur cet exemple, que les fonctions sur les nombres (diviser, additionner..) n'utilisent pas la notation fonctionnelle (identificateur de fonction, suivi des arguments) mais la notation algébrique habituelle (notation "*infixée*"), et que des calculs complexes conduisent à des expressions algébriques avec priorités implicites, conformes à l'usage en mathématiques, et que donc il n'est pas nécessaire d'interpréter comme une composition de fonctions. Par contre, les fonctions sur les chaînes (MID$, LENGTH) utilisent la notation fonctionnelle et la composition de ces fonctions donne des expressions complexes qu'il est avantageux de décomposer comme indiqué ci-dessus. Quand on se limite aux types numériques, les spécificités de la notation fonctionnelle peuvent ne pas apparaître ; nous reviendrons sur ce point au chapitre suivant.

Comme pour l'affectation, l'introduction de la syntaxe fonctionnelle et de la composition des fonctions peut s'appuyer sur des précurseurs en mathématiques et sur des représentations du dispositif.

---

[72] On reprochera peut-être à cette forme de décomposition de trop privilégier l'affectation, et de mettre sur le même plan les variables ayant un sens dans le programme, et des variables intermédiaires ; il serait certainement plus conforme aux principes de structuration de définir une fonction PosiMilieu qui à une chaîne de longueur impaire ferait correspondre la position de son caractère milieu, mais nous plaçons dans un schéma d'apprentissage où les définitions de fonctions viennent dans un second temps.

1.3.1 Les précurseurs en mathématiques de la notation fonctionnelle

Ils ont peu de chances d'être présents chez les élèves que nous observons. Il y a bien eu, dans l'enseignement des mathématiques, à l'école élémentaire puis en classe de quatrième et de troisième des tentatives d'introduction d'un enseignement sur les fonctions incluant la composition des fonctions, et une utilisation de cette composition par exemple pour des problèmes de géométrie. Les fonctions ont été introduites comme opérateurs de transformation (école élémentaire) ou comme cas particulier de relations, elle-mêmes définies par leurs graphes (secondaire). Les difficultés rencontrées dans cet enseignement ont conduit à son abandon. En mathématiques, à la fin du secondaire, les fonctions restent présentes seulement dans l'enseignement de l'analyse; mais il s'agit de fonctions d'une variable réelle, et à valeurs réelles, dont la composition est peu envisagée (les fonctions "inverses" sont vues seulement dans certaines classes en fin de cycle). En géométrie, la correspondance entre points n'est pas envisagée de façon fonctionnelle.

1.3.2 Le traitement d'une écriture fonctionnelle par le dispositif

L'appel d'une fonction suppose que le dispositif soit en un point de l'exécution où il attend une valeur; si cette valeur se trouve résulter de l'appel d'une fonction, le dispositif laisse l'exécution en attente, passe à l'évaluation de l'expression comprenant l'appel fonctionnel; après cette évaluation, le dispositif retourne à l'exécution. Ainsi dans `LET A$=MID$(X$,1,2)` l'instruction d'affectation `LET A$=` est laissée en attente jusqu'à l'évaluation de la fonction `sous-chaîne` avec les arguments `X$, 1` et `2`, puis la valeur résultant de l'évaluation est passée à l'instruction `LET A$=` , et l'instruction est exécutée. L'expression `MID$(X$,1,2)` rencontrée seule provoque par conséquent une erreur de syntaxe: en effet, aucune instruction n'est en attente d'une valeur. S'il y a plusieurs appels de fonctions imbriqués (ce qui correspond à une composition de fonctions), il y a plusieurs niveaux de calculs en attente: dans `LET A$=MID$(X$,LEN(X$),2)` successivement, l'affectation puis la fonction `MID$` sont laissés en attente pendant l'évaluation de `LEN(X$)`. L'état du dispositif varie donc de façon particulièrement subtile au cours de l'interprétation d'une expression contenant un appel de fonction. C'est pourquoi une représentation mentale correcte de l'interprétation d'une écriture fonctionnelle par le dispositif ne peut être considérée comme immédiate pour le débutant

En considérant les essais d'introduction d'un enseignement sur les fonctions dans l'enseignement des mathématiques, on peut prévoir que la rédaction d'un programme comportant des écritures fonctionnelles sera une une tâche difficile pour l'élève. A la différence des mathématiques, les élèves peuvent en informatique, s'appuyer sur une représentation du traitement de ces écritures par le dispositif, mais nous avons montré que cette représentation n'est pas immédiate. Egalement, en informatique, les élèves disposent de l'affectation à une variable intermédiaire qui donne une forme syntaxiquement différente à la composition des fonctions.

1.4 Les actions complexes; l'itération à un point de sortie

1.4.1 Différents types d'actions complexes; notre problématique

Les actions complexes introduisent des ruptures de séquentialité, et donc permettent l'écriture d'algorithmes non-triviaux. Beaucoup d'auteurs font l'hypothèse implicite ou explicite que les difficultés des apprentis-programmeurs apparaissent seulement avec les actions complexes. Nous essayons pour notre part de rechercher les difficultés des élèves dans les spécificités de la programmation pour un dispositif automatique. Si des difficultés apparaissant dans le cadre des actions complexes, nous voulons examiner dans quelle mesure elles se rattachent à des difficultés plus générales pouvant être mises en évidence également dans le cadre séquentiel. Ceci doit nous donner des moyens pédagogiques pour éviter l'accumulation des difficultés qui se traduit souvent par un échec, au moment du passage des actions simples aux actions complexes. Dans cette perspective, nous ne chercherons nullement à être exhaustif dans le domaine des difficultés des élèves concernant les actions complexes. Nous n'aborderons pas la question de la planification des actions dans un projet d'ensemble, tel qu'il est étudié par [HOC 76]: l'article montre en effet qu'une planification correcte

suppose l'existence chez le sujet de Systèmes de Représentation et de Traitement hiérarchisés et l'intériorisation de solutions à des problèmes partiels (traitement algorithmique et représentation des données), alors qu'au niveau où nous situons nos observations, ces éléments sont au mieux en gestation. Il aurait été possible avec nos hypothèses d'aborder la question des alternatives, sous sa forme la plus simple SI <condition> ALORS <actions 1> SINON <actions 2>, puis sous les formes permettant de traiter des conditions multiples (emboîtement, saut conditionnel, structure de cas). Nous verrons ci-dessous que, suite en particulier à la lecture de [ROGALSKI 88] les hypothèses que l'on peut faire sur les difficultés des élèves dans le cadre de l'itération recouvrent assez largement celles que l'on peut faire pour le cadre des alternatives. C'est pourquoi nous avons fait le choix de limiter nos observations concernant les actions complexes à à l'itération à un point de sortie. On trouvera donc ci-dessous une présentation assez complète de l'itération, et de son interprétation à l'aide des assertions, puis une discussion sur les difficultés prévisibles des élèves dans le cadre de l'alternative et de l'itération.

1.4.2 L'itération à un point de sortie

Une itération se construit à l'aide d'un avertisseur de début d'itération, d'une instruction de sortie d'itération et d'un avertisseur de fin d'itération. Elle inclue une (des) alternative(s) incomplète(s) (S I <condition> ALORS <actions> IS) dont le bloc <actions> se termine par l'instruction de sortie d'itération ((EXIT)) FAIRE est l'avertisseur de début d'itération, BOUCLER est l'avertisseur de fin d'itération, EXIT est l'instruction de sortie de l'itération.

```
FAIRE
 A1
 SI <condition1> ALORS
              B1 EXIT IS
 A2
SI <condition2> ALORS
              B2 EXIT IS
.................
 Ak
 SI <conditionk> ALORS
              Bk EXIT IS
 Ak+1
BOUCLER
```

Si l'on considère une itération à un seul point de sortie, il est possible de "sortir de l'itération" le bloc d'actions B_1 de l'alternative unique, et il suffit de deux blocs d'actions (éventuellement vides ou composés d'une seule action) pour composer, avec l'alternative unique, le corps de l'itération. Soient A_1 et A_2 ces deux blocs d'actions: l'itération à un seul point de sortie la plus générale s'exprime :

```
FAIRE
  A1
  SI <condition>
    ALORS EXIT IS
  A2
BOUCLER
```

Nous avons vu ci-dessus (§1.1) comment à chaque action (instruction) d'un programme on peut associer une "pré-assertion", et une "post-assertion" qui se trouve être la pré-assertion de l'action suivante dans le but de donner une signification au programme. La sémantique d'une action A consiste à préciser à quelles conditions sur P et Q, on peut avoir $[P]$ A $[[Q]]$. Caractériser l'itération revient donc à chercher à quelles conditions , une assertion P étant choisie comme pré-assertion de l'itération, l'assertion Q en est une post-assertion

Supposons que l'on ait pu démontrer que l'itération termine et trouver une assertion P telle que

(1) $([[P$ et non <condition>$]]$) A_2 A_1 $[[P]]$,

alors on a ([DIJKRA 76])

$[[P]]$ FAIRE...BOUCLER $[[P$ et <condition>.$]]$

et donc si $P_0 \Rightarrow P$ (P_0 est un cas particulier de P), et $P \Rightarrow Q$. (Q est plus général que P), on a aussi:

$[[P_0]]$ FAIRE...BOUCLER $[[Q]]$

La condition **(1)** exprime **l'invariance** de P par le bloc d'instructions du corps de l'itération. On dit que P est **un invariant de l'itération** ou selon la terminologie de [ARSAC 83], une hypothèse de récurrence. Le premier intérêt d'un invariant d'itération est de prouver que l'itération, si elle termine, opère

effectivement la transformation pour laquelle elle a été construite: en général, l'état final souhaité Q est connu, et donc l'existence d'un invariant P tel que P ET <condition> $\Rightarrow Q$ permet cette preuve. Elle permet également de construire l'initialisation des variables en jeu dans l'itération; certaines valeurs initiales pourront être déduites de l'énoncé du problème, on calculera les valeurs initiales des autres variables de façon que ces valeurs constituent un état P_0 , cas particulier de P . Par exemple, en reprenant l'algorithme d'exponentiation du §1.1, l'invariant d'itération que nous proposons est $r = x^n$, et on doit partir de $n = 0$ de façon que l'algorithme soit valable pour tout exposant entier naturel, et on en déduit la valeur initiale $r = 1$, de façon que $(n = 0 ; r = 1)$ soit un cas particulier de $r = x^n$.

De façon générale un invariant d'itération rend compte de la signification d'une itération; il permet un mode d'interprétation de l'itération différent de l'interprétation comme répétition d'action Faisant l'hypothèse selon laquelle les difficultés de construction d'une itération viennent de ce que le programmeur considère des actions, au lieu de considérer les situations, [ARSAC 80] propose une méthode de création d'itérations, basée sur la production d'un invariant d'itération que nous présentons et discutons dans les annexes à ce chapitre. En particulier, nous confrontons les difficultés observées lors d'études expérimentales auprès de débutants aux hypothèses de J. ARSAC sur ces difficultés, et nous nous demanderons si cette méthode n'introduit pas des difficultés qui dépassent le niveau des acquisitions visées.

1.4.3 L'itération à plusieurs points de sortie.

Les éléments syntaxiques présentés ci-dessus (avertisseurs de début et de fin d'itération, instruction de sortie de boucle) permettent de construire des itérations à sorties multiples. Il est là aussi possible de considérer un invariant d'itération mais la relation entre cet invariant et la post-assertion de l'itération est plus difficile à exprimer. Prenons l'exemple de la recherche du rang de la première occurrence d'un caractère C dans une chaîne CH nonvide, en posant que si C n'a pas d'occurrence dans CH, ce rang sera zéro, et supposons que nous ayons à prouver que le programme suivant est une solution:

```
N← 1
FAIRE
   X← sous-chaîne (CH, N, 1)
   SI X = C ALORS EXIT IS
   SI N = longueur(CH)  ALORS N←0 EXIT IS
   N← N + 1
BOUCLER
```

Nous déduisons de l'énoncé la post-assertion Q de l'itération

(le rang de la première occurrence de C dans CH est N) OU
(C n'a pas d'occurrence dans CH ET N=0) .

De façon à prouver la validité du programme, construisons un invariant d'itération P:

(X est le Nième caractère de CH) ET (il n'y a pas
d'occurrence de C avant le Nième caractère)

Il y a deux conditions de sortie:

<condition 1> = (X=C)
<condition 2> = (X différent de C) ET (N = longueur(CH))

et deux actions

A_1 X← sous-chaîne (CH, N, 1)
A_2 N← N + 1

La condition d'invariance

(1) *[[P* ET non <condition1> ET non <condition2>*]]* A_2 A_1 *[[P]]*,

se vérifie car:

[[il n'y a pas d'occurrence de C avant le Nième caractère*]]* A_1 *[[P]]*,

et donc

[[il n'y a pas d'occurrence de C avant le (N+1)ième caractère*]]* A_2 A_1 *[[P]]*,

Pour que le programme soit valide il suffit que

(2) *P* ET <condition1> ⇒ *Q*

et

(3) *[[P* ET <condition2>*]]* N←0 *[[Q]]*

On vérifie facilement **(2)**

P ET <condition1> ⇒ (le rang de la première occurrence de C dans CH est N) ⇒ *Q*

A cause de la présence de l'affectation N←0 avant l'instruction EXIT, **(3)** est moins simple à établir. Si l'on remplace dans *Q* toutes les occurrences de N par 0, on obtient l'assertion (C n'a pas d'occurrence dans CH) et donc d'après la caractérisation sémantique de l'affectation:

[[R]] N←0 *[[Q]]* si et seulement si *R* ⇒ (C n'a pas d'occurrence dans CH)

et **(2)** est vérifiée car *P* et <condition2> est équivalent à:

(X est le Nième caractère de CH)
ET (il n'y a pas d'occurrence de C dans CH)
ET (N = longueur (CH))

Nous espérons avoir montré avec cet exemple que la présence de plusieurs points de sortie dans une itération multiplie les difficultés d'interprétation, en particulier les assertions font intervenir de nombreux connecteurs logiques. D'autre-part, comme nous le verrons ci-dessous, dans les langages accessibles pour l'initiation, l'écriture d'une itération à plusieurs points de sortie ne se construit en général pas directement. Pour ces deux raisons, les enseignants de l'option informatique n'abordent généralement pas l'itération à plusieurs points de sortie lors de la première année (classe de Seconde). Nous nous conformerons à ce choix, et nous montrerons dans la suite qu'il est possible de construire des problèmes conformes aux objectifs que nous nous sommes fixés impliquant seulement l'itération à un seul point de sortie.

1.4.4 Les cas particuliers d'itération à un point de sortie: forme "tant que"; forme "jusqu'à"

Nous avons indiqué ci-dessus que l'itération à un point de sortie la plus générale peut s'écrire comme ci-contre. Les cas particuliers d'itérations se caractérisent par le fait que soit le bloc d'instructions A_1, soit le bloc d'instructions A_2, est vide. Ainsi,

```
FAIRE
 A1
 SI <condition> ALORS
EXIT                IS
 A2
BOUCLER
```

```
TANT QUE <condition>
                    A
                   QT
```

est une abréviation de

```
FAIRE
   SI <non condition>
      ALORS EXIT IS
   A
BOUCLER
```

```
                REPETER
                    A
      JUSQU'A <condition>
```

est une abréviation de

```
FAIRE
   A
   SI <condition>
      ALORS EXIT IS
BOUCLER
```

Beaucoup de problèmes, en particulier les plus élémentaires, se résolvent à l'aide d'une des deux formes particulières d'itération ci-dessus. Les deux formes se différencient par la place de l'instruction conditionnelle de sortie, par le fait que dans la forme TANT QUE, <condition> est une condition de continuation, alors que dans la forme REPETER, c'est une condition de sortie. De plus, la forme REPETER implique au moins une exécution du bloc d'actions du corps de l'itération, ce qui n'est pas le cas de la forme TANT QUE.[73]

[73] Suivant [ARSAC 90], l'itération à un point de sortie la plus générale
FAIRE A_1 SI <condition> ALORS EXIT IS A_2 BOUCLER

1.4.5 L'élève débutant et les actions complexes

Les 4 articles que nous avons étudié au chapitre 1 analysent l'activité de programmation dans le cadre de ces actions complexes: [GREEN 77] étudie la compréhension des structures alternatives, [SAMURCAY 85] et [COHORS-FRESENBORG 87] situent leurs observations dans le domaine de l'itération, et [HOC 77] étudie l'activité de programmeurs pour un problème impliquant la coordination de diverses formes d'actions complexes. D'autres recherches que nous citons ci-dessous portent sur le même domaine. Nous pouvons nous appuyer sur leurs conclusions pour prévoir les difficultés des élèves face à la conception d'actions complexes, et préciser la problématique qui conduit les observations rapportées dans les chapitres suivants (en particulier au chapitre 6 consacré à l'itération).

De l'étude de T.GREEN, nous déduisons que la structure alternative, malgré son caractère intuitif dans les cas les plus simples, présente de réelles difficultés d'interprétation y compris pour des programmeurs professionnels; l'acquisition d'un style de collecte de l'information, qu'il soit adapté à des alternatives imbriquées ou au style "GOTO" suppose sans doute une longue pratique et des capacités logiques. Dans une étude concernant des débutants. [ROGALSKI 88][74] pointe une autre difficulté: face à une écriture alternative, l'élève interprète le déroulement des instructions selon les compétences qu'il prête au dispositif, et non selon les régles de la séquentialité. On peut demander par exemple à des élèves de décrire l'effet de l'entrée de diverses données pour un programme tel que:

```
AFFICHER 'Etes vous de sexe féminin  ?'
LIIRE REP
SI REP = 'OUI' ALORS AFFICHER 'BONJOUR MADAME' SINON AFFICHER
'BONJOUR MONSIEUR'
```

Beaucoup d'élèves pensent que l'ordinateur ne va rien afficher ou se bloquer si l'on entre par exemple la chaîne `'BOF'`. Ce comportement est à rapprocher des interprétation erronées des actions simples (affectation..) que nous étudierons en détail au chapitre suivant . Il est clair que le contexte des objets en cause a une influence sur ce comportement: la condition `REP= 'OUI'` a, pour ces élèves, un statut beaucoup plus flou que, par exemple, une comparaison sur des entités numériques à laquelle les apprentissages mathématiques antérieurs ont donné un contenu précis.

J. ROGALSKI montre également que des élèves se construisent une compréhension erronée de la réponse du dispositif, mais fonctionnant dans la majorité des cas rencontrés: par exemple, une régle implicite souvent présente chez les élèves est: «*dans la partie conditionnelle d'un programme, quand la condition* `A` *a été traitée, les autres instructions s'appliquent à la situation* `non A`.» Une telle régle erronée peut ne pas être mise en défaut, et donc s'installer durablement chez l'élève si les situations de programmation la valident, par exemple si l'alternative est rencontrée exclusivement comme instruction de sortie d'itération ou de procédure. L'absence d'acquis en logique chez les élèves rend problématique l'écriture et la transformation de la partie <condition> de l'alternative et interagit avec les difficultés évoquées ci-dessus: absence de maîtrise de la négation, non prise en compte de la combinatoire des cas.

Concernant l'itération, nous disposons, parmi d'autres, des résultats de [SAMURCAY 85] présenté au chapitre 1, et de [LABORDE BALACHEFF MEIJAS 1985].et [ROGALSKI 84] présentés en annexe à ce chapitre. De façon générale, la difficulté consiste à passer d'une vision globale d'une solution sous forme d'états (par exemple, dans un problème de sommation, au départ, le compteur et la somme valent 0, à l'arrivée, le compteur a pour valeur le nombre de termes à effectuer, la somme est le

---

peut s'exprimer à l'aide de la forme `TANT QUE`, à condition de répéter le bloc d'actions A_1 avant l'itération, et, dans le corps de l'itération, après le bloc A_2 :

A_1 `TANT QUE <non condition>` A_2 A_1 `QT`

Par contre, l'itération à plusieurs points de sortie impose l'utilisation d'une variable supplémentaire pour être codée avec la forme `TANT QUE`.

[74][ROGALSKI 88] J.ROGALSKI Acquisition de structures conditionnelles (Annales de Didactique et de Sciences cognitives IREM de Strasbourg 1988) .

résultat de l'addition des termes), à l'explicitation d'un ensemble séquentialisé de transformations. Ce passage comporte des points obligés, dont le franchissement constitue, dans les observations rapportées, des difficultés significatives:

- **nécessité de dégager les actions qui constituent le "corps de l'itération"**
- nécessité de concevoir l'instruction conditionnelle de sortie.
- nécessité d'articuler les différentes instructions du corps de l'itération, en particulier, de les ordonner et de les coordonner avec l'instruction conditionnelle de sortie.

Le traitement des variables révèle des difficultés spécifiques à l'itération:

- nécessité d'utiliser des variables dont la signification est invariante au cours du traitement, mais dont la valeur évolue; si le "compteur d'itération" est bien accepté comme une variable de ce type, une difficulté apparaît pour des variables cumulatives telles que la somme intermédiaire dans les différents problèmes.
- l'initialisation de ces variables est une difficulté majeure pour une partie importante des élèves.

On peut s'attendre à ce que les élèves rencontrent des difficultés cohérentes avec celles repérées dans l'acquisition de la structure alternative:

- absence de maîtrise des relations logiques, ce qui crée des difficultés pour l'écriture de la condition d'arrêt et son articulation avec le corps de boucle: en particulier, le programmeur est obligé de considérer à la fois la condition de sortie (qui est réalisée après la sortie de la boucle), et sa négation (qui est réalisée à l'intérieur de la boucle). On sait que, en classe de Seconde, la négation d'une conjonction en disjonction, la négation d'une inégalité stricte en inégalité large n'ont pas fait l'objet d'un enseignement formel, et ne sont pas acquises.([ROGALSKI 88]).
- intervention de représentations erronées des capacités de contrôle du dispositif. [SAMURCAY 85] cite par exemple le cas d'élèves rédigeant la condition de sortie d'une itération ayant pour but de calculer le produit de deux nombres par addition itérée sous la forme suivante: `UNTIL a * b`

Nous pouvons faire l'hypothèse que ces deux difficultés seront d'autant plus présentes que les objets sur lesquels porte l'itération ne seront pas apparus antérieurement sous forme de représentations calculables, et que donc, les élèves ne pourront attacher une signification précise à une condition construite sur ces objets. Par exemple, si, dans un problème de recherche d'une première occurrence, la condition de sortie porte sur la valeur d'une variable booléenne `trouvé`, des élèves pourront considérer que au cours de l'exécution, cette variable possède en elle-même la valeur cohérente avec le point atteint dans l'exécution, et donc ne pas comprendre la nécessité d'une instruction de calcul lui donnant une valeur dans le corps de boucle. Cette hypothèse va dans le sens de nos hypothèses générales sur la mise en évidence des difficultés des élèves à partir de problèmes où existe une certaine distance entre un mode de résolution connu des élèves dans les objets du monde, et le mode de résolution pour un dispositif automatique. Elle se place dans la recherche d'une cohérence entre les difficultés constatées dans la construction de problèmes sans rupture de séquentialité et celles qui apparaissent avec l'itération. Elle implique de faire travailler les élèves sur des itérations sollicitant fortement une structure non-numérique, ce que nous ferons dans les études expérimentales rapportées aux chapitres **6**, **7** et **9**.

Nous avons présenté et discuté en annexe, une méthode de construction d'itération proposée dans [ARSAC 80] et [ARSAC 83], dont l'influence s'exerce notablement dans l'option informatique des lycées. Il nous semble que, comme pour l'affectation, une explicitation logique complète de l'itération dépasse nettement le niveau d'acquisition visé, et que la construction de l'itération par repérage d'une régularité dans une suite d'action constitue un premier niveau d'acquisition même s'il ne résout pas tous les problèmes (en particulier celui de la preuve et de l'initialisation). De même, nous nous demandons si la distinction entre répétition et itération est réellement convaincante. En principe la répétition devrait être une première forme d'itération s'interprétant sans l'invariant d'itération, mais certains calculs itératifs classés en répétition nous on paru s'interpréter plus difficilement que d'autres classés en itération.

2. Les chaînes de caractères

Elles constituent la première structure de données que nous allons étudier, comme support de la codification au cours de l'étude expérimentale des parties II et III. Nous présentons au paragraphe 2.1 une définition de la structure chaînes de caractères : cette définition se fait par un ensemble réduit de fonctions que nous considérons comme suffisant, à la fois d'un point de vue théorique, et d'un point de vue pédagogique. Elle nous permet de commencer à situer les relations entre le débutant en programmation et la structure de chaîne (§ 2.2) et de faire des hypothèses préparatoires à l'étude expérimentale des parties II et III (§ 2.3)

2.1 Définition de la structure

Nous nous intéressons aux fonctions définissant le type, ainsi qu'aux types réutilisés; nous donnerons quelques propriétés usuelles des fonctions, mais, nous ne chercherons pas à présenter un système d'axiomes.

2.1.1 Les types réutilisés

- le type caractère: il s'agit des éléments d'un alphabet, qui peut être précisé par le contexte. Il n'y a pas ici de difficulté majeure, si ce n'est d'accepter que des caractères "spéciaux" tels que l'espace, ou le retour de chariot font partie de l'alphabet.
- le type entier, qui se subdivise, en fait en type *ordinal* et type *cardinal*; nous verrons plus loin que les difficultés posées par cette double structure numérique sont nombreuses.

2.1.2 Les fonctions primitives

Le jeu réduit de fonctions suivant est suffisant pour définir la structure "chaînes de caractères":

- `chaîne constante`: à une suite ordonnée de caractères (éventuellement vide), on associe une chaîne; la notation consiste à écrire les caractères sous leur forme externe (celle qui est affichée à l'écran), la suite étant encadrée par des caractères conventionnels: par exemple `'azer ty(§)er'` est une chaîne en PASCAL.
- `longueur`: à une chaîne on associe un nombre qui peut être considéré comme un *cardinal* (le nombre des caractères de la chaîne) ou un *ordinal* (la position du dernier caractère de la chaîne); la fonction ayant un seul argument, la notation fonctionnelle à une variable est généralement employée.
- concaténation: à deux chaînes on associe une chaîne. Le résultat est la suite de caractères formée de la suite des caractères du premier argument, suivie de la suite des caractères du second argument. La fonction étant à deux arguments, on peut utiliser la *notation fonctionnelle à deux arguments*, par exemple concat(chaîne1, chaîne2), ou la *notation infixée* comme dans les opérations arithmétiques, le signe opératoire étant, par exemple !: 'xyz' ! 'abcd' a pour valeur 'xyzabcd'. La notation infixée et l'associativité permettent une écriture aisée de concaténations composées: 'abcd' ! 'efgh' ! 'ijkl' a pour valeur 'abcdefghijkl'
- sous-chaîne: rend une chaîne; ses arguments sont une chaîne, un ordinal (marquant le rang du premier caractère du résultat dans la chaîne argument), et un cardinal égal au nombre de caractères du résultat.
- Les **opérateurs d'égalité et de comparaison à valeur booléenne** opèrent sur les chaînes de caractères. L'égalité de deux chaînes suppose qu'elles aient exactement les mêmes caractères; par exemple, une chaîne composée d'espaces n'est pas égale à la chaîne vide. Les opérateurs de comparaison (stricte ou large) opèrent selon l'ordre alphabétique étendu aux caractères non alphabétiques par la relation d'ordre impliquée par la numérotation en machine des caractères; ainsi, les chiffres sont dans l'ordre croissant, avant les majuscules, qui précédent elle-même les minuscules. On a 'RAT' > 'CHAT', et '4' > '32' !
- Les **fonctions de conversion de type** associent à un nombre une chaîne de chiffres constituant son écriture en numération de position dans une base de numération (généralement la base décimale), et réciproquement.

2.1.3 Les autres fonctions

- La fonction `position` est souvent implémentée dans les langages; ayant deux arguments `chaîne1` et `chaîne2`, elle rend la position du premier caractère de la première occurrence de `chaîne1` dans `chaîne2`, si cette occurrence existe, sinon 0; elle se construit à l'aide du jeu restreint ci-dessus, et d'une itération ou d'une récursion, par exemple comme ci-dessous (le résultat de la fonction est la valeur de la variable `N`).

```
N← 1
L← longueur(chaîne1)
FAIRE
   X← sous-chaîne (chaîne2, N, L)
   SI X = chaîne1 ALORS EXIT IS
   SI N = longueur(chaîne2)  ALORS N←0 EXIT IS
   N← N + 1
BOUCLER
```

  Cette fonction est évidemment fort utile pour une recherche d'occurrence ou une recherche d'appartenance. Nous ne la mettons cependant pas au même niveau que le jeu précédent, car sa construction implique une forme d'itération ou de récursion (qui peut très bien être différente de celle que nous présentons ci-dessus), et sans entrer dans des considérations de complexité, utiliser cette fonction comme **primitive** conduirait à concevoir le jeu de fonction comme un tout indistinct, masquant le fait que certaines fonctions peuvent être construites, un peu comme si, dans une théorie mathématique, on présentait de façon indistincte **axiomes** (ou propositions initiales) et **théorèmes**[75]. En particulier, il est tout à fait intéressant que les élèves puissent **construire cette fonction en réponse à un problème** avant même de la considérer dans sa généralité. Ainsi, la construction d'une fonction pourrait être comparée à la démonstration d'un théorème en mathématiques : le théorème est d'abord une connaissance locale fonctionnant dans le cadre d'un problème, et des éléments de déduction interviennent dans ce cadre, avant que le théorème soit considéré et démontré dans sa généralité.
- D'**autres fonctions** peuvent exister dans les langages; elles constituent en fait le plus souvent des cas particuliers d'utilisation des fonctions ci-dessus, et ne présentent pas d'intérêt pour notre propos.

2.1.4 Les structures numériques

Deux structures numériques [76]sont impliquées dans les chaînes de caractères:

---

[75]Il serait bien sûr possible de définir une structure de chaîne où la fonction `position` serait **primitive**, et la fonction `sous-chaîne` **construite par itération** ; il s'agirait alors d'une organisation associative des données (accessible par le contenu) ; on aurait une structure différente, bien que partageant avec la structure de chaîne les fonctions avec les mêmes propriétés.

[76]Plus généralement, on peut considérer qu'un même type d'objet peut être abstrait sous deux formes : une forme statique considérant l'ensemble comme déjà achevé, et une forme générative dont les fonctions permettent la construction de chacune des valeurs, à partir d'un nombre fini de valeurs. Il est clair que la forme statique de l'entité "entier" est la structure cardinale, et sa forme générative est la structure ordinale. La structure abstraite que nous avons donnée pour les chaînes de caractères (fonctions sous-chaîne, longueur, concaténation et prédicat égalité) est de la forme statique ; la forme générative correspondante serait en fait celle d'une pile de caractères définie par exemple par les fonctions `chaîne-vide` , `met-à-droite` ((chaîne, caractère) → chaîne), `ote-à-droite` (chaîne → chaîne), `dernier-caractère` (chaîne → caractère), et le prédicat `chaîne-vide?` Dans l'activité de programmation, le passage d'une forme à l'autre, et donc leur coordination peuvent être cruciaux : ainsi les chaînes de caractères sont considérées sous leur aspect statique dans les premiers traitements (problèmes non itératifs, ou parcours itératif d'une chaîne), mais les problèmes impliquant la construction itérative ou récursive d'une chaîne (par exemple, le problème classique de retournement d'une chaîne) obligent à considérer la forme générative. Nous n'examinerons pas les conséquences de ce point de vue concernant les chaînes, car nous nous limitons à l'observation des premiers traitements, mais il nous permet d'apercevoir que les difficultés de coordination ordinaux-cardinaux pourraient être examinées comme cas particulier de difficultés à coordonner les structures statiques et génératives d'une

- les cardinaux, c'est à dire la structure classique de l'ensemble des entiers muni des deux opérations.
- les ordinaux, c'est à dire l'idée intuitive du premier, du second..., ou de façon mathématique, un ensemble Ω muni d'une application injective (successeur) , et d'un élément a_0 (premier élément) tels que:
    - le premier élément a_0 n'est le successeur d'aucun élément,
    - si une partie de Ω contient le premier élément a_0 et les successeurs de tous ses éléments, alors elle est égale à Ω.

2.2 Les chaînes de caractères pour l'élève débutant

2.2.1 Les représentations spontanées des chaînes

On sait peu de chose de la façon dont les débutants considèrent les objets informatiques, et comment leur représentation de ces objets intervient dans leur activité de programmation. Par rapport à d'autres objets, les représentations spontanées des chaînes de caractères (c'est-à-dire les représentations qui interviennent au début de l'apprentissage) vont être, à notre avis, influencées par trois particularités:

- les chaînes présentent une **forme externe facilement manipulable**: on peut faire afficher, et entrer (lire au clavier) des chaînes de caractères. Il suffit de considérer les booléens ou les tableaux pour constater qu'il n'en est pas de même pour tous les types. D'autre part, le codage en machine est relativement conforme à cette forme externe (suite ordonnée de caractères), et introduit peu de limitation, ce qui n'est pas le cas, par exemple, des types numériques (la forme externe décimale introduit une approximation, et donc, deux nombres peuvent avoir la même forme externe sans avoir la même valeur interne). La forme externe donne donc une idée des chaînes qui n'est pas trop éloignée du codage interne.
- par contre, la **représentation de l'organisation mutuelle interne** des diverses chaînes peut être mystérieuse: quand on réalise l'affectation `x← y + 1`, ou même `x← y` , il est clair que les valeurs de `x` et `y` sont conservées de façon distincte; les valeurs sont *duplicables.*, et donc une modification de `y` n'a pas d'influence sur `x`. Par contre, après l'affectation `x← sch(y, n, p)`, on peut se représenter l'organisation en mémoire de façon que `x` partage avec `y` physiquement les mêmes caractères, et imaginer que donc, une modification sur l'une des variables ait une influence sur l'autre. Cette organisation en mémoire est effectivement possible, mais la valeur de la chaîne est un alors un pointeur et non l'ensemble des caractères, ce qui empêche qu'une modification sur une des variables ait une influence sur l'autre,
- si l'on considère des problèmes réels (c'est-à-dire non réduits à des manipulations à l'intérieur des objets du langage), les objets manipulés peuvent être **porteurs de propriétés qui n'existent pas dans les chaînes de caractères**; par exemple, une phrase est structurée en mots et en propositions, les régles syntaxiques permettant cette structuration pouvant difficilement être produites dans leur totalité: trouver un mot donné dans une phrase met en jeu des mécanismes de recherche complexes où le sens qui est donné aux éléments de la phrase intervient; une hypothèse est donc que, dans leurs représentations spontanées, les élèves, dans le cadre de problèmes réels, prêtent au dispositif des capacités de traitement de chaîne qui sont celles d'un opérateur humain. Dans le même ordre d'idée, les débutants peuvent éprouver des difficultés à donner une signification aux opérateurs de comparaison (égalité, ordre alphabétique) ou leur attribuer des significations issues du contexte courant.

2.2.2 Les difficultés et enjeux cognitifs relatifs aux fonctions

- **les fonctions "`longueur`" et "`concaténation`"**: la fonction `longueur` est un morphisme de l'ensemble des chaînes muni de l'opération `concaténation` vers l'ensemble des entiers positifs muni de l'addition; elle possède les mêmes

---

même entité.

propriétés que par exemple la fonction `cardinal` pour la réunion disjointe des ensembles: si deux ensembles `A` et `B` sont disjoints, `card(A U B) = card(A) + card(B)`

On peut donc raisonnablement penser que les élèves vont opérer des transferts à partir de leur propre connaissance des structures additives; en effet le rapport entre addition des cardinaux et réunion disjointe a été fait dès le cours préparatoire (5 à 6 ans), puis les extensions successives des ensembles de nombres ont conduit l'élève à associer l'addition des nombres (puis de vecteurs et des angles) à des opérations sur des grandeurs physiques.

D'autre part, l'écriture des fonctions a déjà été rencontrée: la notation infixée qui peut être utilisée pour la concaténation est employée pour les opérations depuis les premiers apprentissages. La notation fonctionnelle à une variable (utilisée pour la fonction `longueur`) a été rencontrée dans les apprentissages mathématiques au collège dans les années qui précédent la classe de seconde: pour noter les fonctions numériques, telles que les polynômes, mais aussi les fonctions sur des ensembles non numériques, et dont la valeur est une grandeur (longueur d'un segment...).

Retenons cependant comme une difficulté possible que, en tant que fonction ayant des chaînes à la fois comme argument et comme résultat, la concaténation peut être comprise comme une "action" transformant les chaînes arguments, indépendamment du résultat rendu; il peut en effet y avoir un doute sur la façon dont la mémorisation des chaînes est opérée dans le dispositif: sont-elles des entités concrètes qu'il faut déplacer effectivement pour les concaténer, où sont-elles dupliquées chaque fois qu'une fonction opère sur elle?.

- **la fonction "`sous-chaîne`"**: elle présente plusieurs particularités qui constituent une rupture avec les connaissances antérieures des élèves sur les fonctions:
    - il s'agit d'une fonction à plusieurs arguments hétérogènes (la notation fonctionnelle à une variable, comme la notation infixée à deux variables ne peuvent s'employer);
    - le résultat n'est pas numérique; si l'on peut penser que les élèves ont travaillé sur des fonctions dont l'argument n'est pas numérique (nombre d'éléments d'un ensemble, longueur d'un segment...), par contre, les apprentissages sur des fonctions dont le résultat n'est pas numérique, concernent essentiellement les transformations géométriques, qui sont très éloignées des fonctions que nous étudions ici.
    - parmi les arguments, les deuxième et troisième sont de type numérique; mais il ne s'agit pas en fait du même type: l'un est un **ordinal**, l'autre un **cardinal**.
    - d'un point de vue conceptuel, `p` et `n` étant fixés, la fonction `chaîne→sous-chaîne(chaîne,p,n)` est une application de l'ensemble des chaînes vers l'ensemble des chaînes de longueur inférieure ou égale à `n`; les seules fonctions connues des élèves, auxquelles une telle application pourrait de ce fait, s'apparenter, sont les projections en géométrie; on remarquera cependant que ses propriétés sont fort éloignées de celles des projections: l'ensemble des chaînes n'est ni un espace vectoriel ni un espace affine, et sauf dans le cas `p=1` , l'application n'est pas idempotente[77]. Il n'y a donc rien, dans les acquisitions antérieures, qui permette de préparer l'acquisition d'une telle fonction.

  Nous verrons au chapitre suivant comment ces particularités interférent avec des représentations erronées du dispositif, et les formulations en langage naturel pour produire une compréhension de "`sous-chaîne`" comme une "action" sur la chaîne argument, ne produisant pas de résultat, ce qui constitue une difficulté majeure.
- la **fonction `position`**: nous avons indiqué pourquoi nous pensons que cette fonction ne peut faire partie du jeu de fonctions primitives; par contre, il est intéressant qu'une telle fonction soit construite par les élèves comme réponse à un

[77] On dit qu'une application `U` est idem-potente si `U` composée par elle-même est égale à `U` ; l'idempotence caractérise les projections parmi les applications affines.

problème à l'aide des fonctions primitives ("recontextualisation"), et en lien avec l'acquisition de l'itération, avant d'être employée comme primitive du langage ("redécontextualisation").

- **les fonctions de conversion**: le principal problème cognitif posé par ces fonctions est bien évidemment leur nécessité, le débutant confondant généralement un nombre avec sa forme externe; nous utiliserons ceci comme élément de rupture conduisant à mettre en cause la conception erronée des objets que sous-tend cette confusion entre entité chaîne et entité numérique.

2.2.3 Les relations entre les deux structures numériques

L'acquisition des structures numériques cardinale et ordinale se fait dès les premières années d'enseignement primaire (Cours préparatoire en France); la question des relations entre les deux structures est essentielle à ce niveau, et conditionne l'acquisition de la notion de nombre. (Voir à ce sujet [PIAGET SZEMINSKA 67][78]). Par la suite, l'enseignement des opérations et des grandeurs concerne exclusivement la structure cardinale, et ses extensions successives au continu (fractions, décimaux, réels...). Egalement, la notation algébrique (c'est à dire l'introduction de lettres dans les calculs, représentatives de quantités) concerne seulement des quantités apparentées aux cardinaux (entiers, réels, vecteurs...). Beaucoup plus tard, après l'enseignement obligatoire, la structure ordinale reparait dans les apprentissages, mais plutôt comme savoir technique: les indices des suites de l'analyse sont des ordinaux, et les changements d'indices, par exemple dans les sommations, sont des correspondances entre ordinaux, pour lesquelles les étudiants peuvent rencontrent des difficultés.

Il nous semble que l'enseignement de l'informatique repose la question des relations entre la structure cardinale et la structure ordinale; voyons-en trois aspects:

- **une même variable peut être de type ordinal ou cardinal** selon l'expression où elle intervient (ce qui constitue en fait une conversion de type implicite): considérons les expressions suivantes (où `chaîne` est une variable de type "chaîne de caractères".), et `n` une variable entière dont la valeur est comprise entre 1 et `longueur(chaîne)`

```
expression1: sous-chaîne(chaîne,1,n)
expression2:  sous-chaîne(chaîne,n,1)
```

  `expression1` peut être interprétée comme la sous-chaîne constituée des `n` premiers caractères de chaîne; `n` doit être considéré dans cette expression comme le **nombre** (cardinal) de caractères dans la sous-chaîne. `expression2` peut être interprétée comme le `n`-ième caractère de la chaîne; `n` doit être considéré dans cette expression comme la **position** (ordinale) de `expression2` dans `chaîne`.
  Notons que cette double structure des entiers est assez générale en informatique: un compteur d'itération repère de façon ordinale le rang du passage dans la boucle en cours, mais peut aussi, si le premier pas d'itération est numéroté 1, repérer le nombre (cardinal) de passages dans la boucle effectués. En ce qui concerne les élèves, on peut faire l'hypothèse que le manque de familiarité avec les ordinaux, entraînera une compréhension des entiers prioritairement comme cardinaux, et donc des difficultés avec le traitement des arguments d'une fonction telle que "`sous-chaîne`".

- **l'addition entre un ordinal et un cardinal**:
  considérons le programme suivant où il s'agit, une lettre majuscule étant entrée, de chercher par correspondance dans la chaîne `alphabet`, la minuscule correspondante:

```
alphabet←'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwx
yz'
afficher 'numéro de la lettre ?'
lire n
```

---

[78] J. PIAGET A.SZEMINSKA "La genèse du nombre chez l'enfant" Delachaux Niestlé, chapitre VI : "L'ordination et la cardination".

```
afficher 'Majuscule (M) ou Minuscule (m) ?'
lire rep
si rep='M' alors resultat← sous-chaîne(alphabet,n,1)
                 sinon resultat←
sous-chaîne(alphabet,n+26,1)
afficher resultat
```

Que signifie l'addition `n + 26` dans:

```
sous-chaîne(alphabet,n+26,1)  ?
```

Il s'agit d'une addition externe, entre un ordinal (`n`), et un cardinal (`26`, le nombre de lettres de l'alphabet). Mathématiquement cette addition s'apparente à l'addition à un point (d'un espace affine) d'un vecteur (de l'espace vectoriel associé). C'est-à-dire que la fonction `n→n+26` est une *translation*. On dispose de la translation inverse `n→n-26` qui donne un sens à la soustraction d'un cardinal à un ordinal. Dans la pratique de l'informatique, la translation et ses propriétés sont utilisées très naturellement: si l'indice de départ, par exemple d'un tableau, est modifié, on modifie en conséquence les autres indices par addition d'un entier; on connait également les programmes assembleurs "*translatables*" dont toutes les adresses sont données par référence au compteur ordinal: l'adresse absolue (ordinale) est obtenue comme somme du compteur ordinal et de l'adresse relative (cardinale). La translation donne par ailleurs un sens à la soustraction de deux ordinaux: `x` et `y` étant deux ordinaux, `x-y` est le cardinal `n` tel que `x =-y + n`.
A ce niveau, les élèves ont juste commencé à aborder le vectoriel, et ses relations avec l'affine; la compréhension de l'addition et de la soustraction des entiers sera très fortement influencée par l'addition et la soustraction des cardinaux; on peut prévoir des difficultés quand seule l'interprétation en terme d'ordinaux donne un sens aux expressions.

- l'**homothétie affine** définie comme fonction `(ordinal, cardinal) → ordinal`
  Considérons le programme suivant, où il s'agit d'accéder de façon directe à des mots ayant tous la même longueur, et rangés dans une chaîne.

```
chaîne←
'LUNDI   MARDI   MERCREDIJEUDI   VENDREDISAMEDI   DIMANCHE'
afficher 'Numéro du jour de la semaine (1 à 7) ?'
lire N
resultat←sous-chaîne(chaîne, 1 + 8*(N-1), 8)
```

  Nous considérons l'application `N → 1 + 8*(N-1)` comme une *homothétie* de centre 1 et de rapport 8. De telles "homothéties" sont employées dans la pratique de l'informatique pour le calcul par exemple de l'adresse de début d'un élément d'un tableau, connaissant l'indice de l'élément. On peut penser que les élèves ont rencontré l'homothétie en géométrie, et que par ailleurs, ils ont travaillé sur des fonctions "affines" dans le domaine numérique (on dit qu'une fonction numérique `f` est affine s'il existe deux nombres `a` et `b` tels que `f(x) = a x + b`). La question qui est posée, est de savoir quels transferts peuvent se faire des compétences dans ce domaine, vers l'acquisition des opérations sur ordinaux et cardinaux nécessaires en informatique.

2.3 Conclusion: travailler sur les chaînes de caractères avec des élèves débutants

En abordant les chaînes de caractères avec les élèves, notre intention est de leur faire rencontrer les difficultés liées à la distance entre le traitement dans les objets du monde et le traitement par un dispositif informatique, en faisant l'hypothèse qu'il pourront surmonter ces difficultés et préparer ainsi des acquisitions ultérieures en informatique.

2.3.1 Difficultés attendues et points d'appui

- la fonction `sous-chaîne` combine plusieurs particularités non encore rencontrées par les élèves; on peut s'attendre à des difficultés dans l'acquisition d'une telle fonction, en particulier, la compréhension erronée comme "action" opérant sur des

entités concrètes en les modifiant.

- les élèves peuvent s'appuyer sur leurs connaissances des structures additives pour l'acquisition des fonctions `longueur` et `concaténation`; par contre, la concaténation opérant sur des chaînes pourrait, être comprise comme modifiants ses arguments, de même pour la fonction `sous-chaîne`.
- l'acquisition des fonctions sur les chaînes de caractères repose la question de l'acquisition de la structure ordinale, et de ses rapports avec la structure cardinale; en effet, les élèves n'ont pas travaillé sur les quantités ordinales dans leurs études récentes, et la notation algébrique a été introduite seulement sur les quantités cardinales.

2.3.2 Validité des acquisitions sur les chaînes de caractères

L'acquisition des chaînes de caractères concerne les fonctions à plusieurs variables, les fonctions dont le résultat n'est pas numérique, les ordinaux et leurs rapports avec les cardinaux. En construisant de nouvelles fonctions à l'aide de primitives (nous avons donné l'exemple de la fonction `position`) les élèves peuvent apercevoir comment une structure de données se construit. Il nous semble donc qu'un travail sur les chaînes de caractères, fait dans cet esprit, prépare des acquisitions ultérieures en informatique. Des compétences acquises par un travail sur les chaînes de caractères constituent donc des compétences de base en informatique. Ceci ne veut dire que "avec les chaînes de caractères on peut tout faire en informatique": la note historique en annexe à ce chapitre (annexe **5 §5**) montre que la puissance d'expression des chaînes de caractères ne s'est pas pas montrée suffisante pour certaines classes de problèmes, même avec un jeu de fonctions étendu. C'est pourquoi, pour nous, un travail sur les chaînes visant à l'acquisition du jeu complet de fonctions existant dans un langage (voir par exemple ci-dessous le cas de LSE en annexe **5 § 4.3**) n'irait pas dans le sens de nos hypothèses.

3. Utilisation des langages de programmation réellement existants

Nous faisons, en annexe à ce chapitre (annexe **5 § 4**) une présentation de plusieurs langages parmi ceux généralement utilisés pour l'initiation. Nous présentons en particulier les trois langages autorisés lors de l'épreuve terminale de l'option informatique: il s'agit de BASIC (**§4.1**), LSE (**§4.2**) et PASCAL (**§4.3**). Ces trois langages présentent une structure algorithmique impérative, et donc conforme à la présentation que nous avons faite au sous-chapitre **1.** Ils diffèrent cependant par l'existence en BASIC d'une organisation en lignes, et d'une instruction de branchement pour la réalisation d'actions complexes: en particulier, les alternatives en chaînes et l'itération doivent se réaliser par branchements. PASCAL impose par contre des formes "*structurées*" d'itération, et l'imbrication des alternatives. On peut considérer que LSE occupe une position intermédiaire, puisque ce langage dispose de formes structurées d'itération, et d'une possibilité partielle d'imbrication des alternatives. Les trois langages distinguent le type chaîne de caractères des autres types (ce n'est pas le cas de LOGO), et emploient des formes variées de déclaration du type des variables. LSE et BASIC disposent en standard des fonctions de traitements que nous avons présenté ci-dessus (**2.1.2**), et des fonctions `position` et de conversion de type (**2.1.3**). Selon les langages et les implémentations, la gestion peut être statique (les chaînes occupent une place constante) ou dynamique (la place mémoire des chaînes est déterminée à l'exécution, et donc peut être aussi grande que la mémoire le permet); il est possible que les limitations dues à une gestion statique n'apparaissent pas dans les problèmes posés au niveau de l'initiation. Bien que PASCAL considère les chaînes comme des tableaux, des versions courantes de PASCAL possèdent le jeu de fonction. Les trois langages disposent de fonctions supplémentaires; c'est particulièrement le cas de LSE. Nous avons indiqué que pour nous, l'enjeu d'un apprentissage sur les chaînes est plutôt la construction de ces fonctions supplémentaires en réponse à des problèmes, que leur acquisition telle quelle. La structure chaîne de caractères existe également dans des langages algorithmiques (**Aladin**, **Medee**, **Pegase...**), ou des outils de

traitement spécialisé des données (Tableurs, S.G.B.D.); le jeu restreint de fonctions que nous avons proposé est présent dans ces langages, la structure algorithmique étant impérative, et conforme à ce que nous avons décrit au § 1 de ce chapitre, à l'exception du tableur. A l'exception du tableur qui demanderait une analyse particulière, il n'y a pas d'obstacle à la transposition dans ces langages de la démarche que nous mettons en oeuvre dans les chapitres suivants.

Analysant la structuration des données "textuelles" à l'aide de listes et de mots dans le langage LOGO (annexe **5 §, 4.5**), nous avons montré que cette structuration conduit à des raisonnements supposant un fonctionnement mental différent de celui qui est nécessaire pour un parcours itératif de chaîne de caractères. le traitement de liste impose d'emblée un niveau de raisonnement équivalent au raisonnement par récurrence, alors que l'acquisition de ce niveau de raisonnement est, en initiation, un objectif, et non un pré-requis. Nous avons vu également que le parcours itératif d'une chaîne se prête mieux à l'interprétation comme action répétée, et peut s'appuyer sur des représentations mentales plus proches d'un dispositif matériel.

Conclusion de la partie I

Ici se termine la première partie de cette thèse. Nous avons, pour présenter notre problématique, d'une part tiré parti de lectures d'ouvrages d'informatiques et de comptes rendus de recherches en psychologie de la programmation et d'autre part exposé nos choix et hypothèses.

1. Les structures de données, les représentations mentales

Nous avons vu au premier chapitre qu'une structuration des données se définit en informatique sous le double aspect *abstraction-représentation*: le *type abstrait* est un ensemble de définitions algébriques de fonctions ; une *représentation* de ce type dans un autre intervient au moment de la programmation effective pour un langage de programmation donné; la *représentation* constitue également un lien abstrait entre types. Les définitions abstraites permettent de considérer les structures de données sous leur forme générale et donc d'éviter les constructions pédagogiques artificielles qui résulteraient d'une conception des données trop liée à un langage ou à un dispositif donné : la concurrence entre langages de programmation a en effet souvent conduit les concepteurs d'un langage à présenter de façon primitive un jeu très étendu de fonctions pour une structure donnée, et donc le pédagogue, si son enseignement est trop fortement lié à un tel langage, peut être tenté de se fixer comme objectif l'assimilation par l'élève de ce jeu complet ; au contraire, le fait de considérer la définition abstraite lui permet de fixer au départ un jeu réduit de fonctions de base, indépendant du langage à l'intérieur d'une classe donnée de langages, les autres fonctions devant être construites par les élèves à partir de ce jeu réduit, dans le cadre de la résolution de problèmes de programmation.

Pour l'élève, l'utilisation d'une structuration des données est comparable au raisonnement à l'intérieur de structures abstraites en mathématiques (par exemple, en géométrie, le raisonnement à partir des hypothèses et propriétés générales, et non à partir des évidences de la figure); la représentation d'une structure dans une autre s'apparente, quant à elle, aux isomorphismes de structure en mathématiques (par exemple, l'isomorphisme entre le corps des nombres complexes et l'ensemble des couples de nombres réels). Ceci permet de situer le niveau conceptuel de l'utilisation des structures de données, et de mettre en relation les apprentissages liés à la structuration des données avec des apprentissages en mathématiques : comme en géométrie, le sujet abordant l'utilisation d'une structuration des données doit apprendre à raisonner sur les propriétés abstraites de la structure, et non sur les propriétés intuitives des objets que la structuration des données modélise; comme dans l'utilisation des nombres complexes, il doit concevoir que la structure de donnée abstraite qu'il utilise est susceptible de plusieurs représentations, déterminer les circonstances où une représentation est nécessaire et savoir choisir la plus adaptée dans un contexte donné.

La première difficulté que peut rencontrer un débutant est donc la nécessité de se dégager d'une conception intuitive des données pour envisager les structures de données sous ce double aspect *abstraction-représentations*. Il pourra d'autre part rencontrer des difficultés plus techniques : la représentation calculable, nécessaire pour la programmation, peut ne pas être adaptée à l'interaction entre le programme et l'utilisateur, par exemple il existe, dans les langages de programmation, des types calculables, mais n'ayant pas de représentation externe (les booléens, les types énumérés...), d'où la nécessité de différencier les représentation d'un même type, et d'apprendre des procédures de passage d'une représentation à l'autre.

Nous avons vu que la recontextualisation des concepts informatiques peut s'opérer dans des stratégies de résolution des problèmes où interagissent les structures de données et les algorithmes (à la différence de stratégies fondées sur une structuration "naturelle" et par conséquent figée des données) : dans cette interaction, il semble intéressant de jouer sur le double aspect *abstraction-*

représentations, un choix judicieux de représentation permettant de faire évoluer l'algorithme. De plus, l'"histoire des structures de données" montre qu'elles interviennent d'abord dans une classe donnée de problèmes, avant d'être étudiées pour elle-mêmes ; la recontextualisation peut donc passer aussi par la recherche de situations où une nouvelle structure intervient comme solution dans une classe de problèmes, cette nouvelle structure étant considérée initialement par sa représentation dans une structure connue; un raisonnement plus général sur cette classe de problèmes, par exemple un calcul de complexité, permet à l'élève de "redécontextualiser" son savoir sur cette structure.

Au second chapitre, nous avons vu que, face à un problème de contrôle d'un dispositif, le sujet met en oeuvre des Systèmes de Représentation et de Traitement (**S.R.T.**), lui permettant de se représenter mentalement une tâche effectuable sur un dispositif donné, et de faire fonctionner mentalement ce dispositif ; pour résoudre, il élabore des codes lui permettant de passer d'un **S.R.T.** où il peut se représenter une solution à un **S.R.T.** compatible avec le dispositif. Dans le cas d'un problème de programmation impliquant la différenciation et la coordination de tâches sur plusieurs niveaux, la conception descendante d'une solution est possible seulement si le sujet dispose d'un ensemble de **S.R.T.** organisés hiérarchiquement lui permettant d'envisager une planification des tâches compatible avec le dispositif.

L'enseignement des méthodes, visant à donner des capacités d'analyse d'un problème complexe en vue de la conception d'un programme, ne semble donc pouvoir commencer que lorsque ce degré de maîtrise est atteint; au contraire, l'enseignement à des débutants aurait plutôt pour fonction la constitution de **S.R.T.** d'abord pragmatiques, puis se différenciant et se structurant. Les structures présentes dans le langage ont une influence importante sur le mode de fonctionnement mental (constitutif d'un **S.R.T.**) que le sujet met en oeuvre pour l'interprétation des programmes.

De même, les conditions de constitution des **S.R.T.** peuvent dépendre du niveau du dispositif : un dispositif de bas niveau, comme la machine à registres de [COHORS-FRESENBORG 87] rend possible l'explicitation du langage, et, en programmant pour ce dispositif, le sujet constitue ses représentations mentales en même temps qu'il programme des structures évoluées à partir des structures primitives du langage. Notre étude, faisant intervenir la structuration des données, suppose un dispositif au langage moins élémentaire ; le terrain d'expérimentation que nous projetons implique également l'emploi d'un langage "évolué".

Dans ces conditions, la constitution des représentations mentales est moins claire: les concepts de base prennent leur sens seulement par leur interaction, et l'énoncé a priori des régles du langage est sans intérêt ; le sujet doit donc, à partir des exemples qui lui sont fournis et des effets en retour du dispositif, se construire une "signification" et imaginer un "fonctionnement" de ces concepts. Le plus souvent, quand il s'agit de problèmes de modélisation de situation familières, la recherche d'une solution passe par l'importation de plans, c'est-à-dire de représentations schématiques servant de guide à partir de **S.R.T.** adaptés à cette situation familière, mais les éléments du **S.R.T.** ainsi construit pour la résolution sur le dispositif peuvent rester marqués par les propriétés des objets et traitements dans le contexte familier. Les connaissances antérieures du sujet interviennent comme précurseurs "producteurs" ou "réducteurs".

2. Nos choix et hypothèses

Le but de cette thèse est de contribuer à l'étude des modes d'acquisition des connaissances en informatique. Dans son état actuel, la didactique de l'informatique ne dispose pas de bases fermement établies dans ce domaine. C'est pourquoi, le terrain le mieux adapté pour une recherche est celui des premiers apprentissages en informatique, où les spécificités d'apprentissages antérieurs en informatique n'interviennent pas et où l'hétérogénéité permet d'observer des élèves qui rencontrent effectivement des difficultés et qui, par conséquent, ne poursuivent généralement pas au delà. L'enjeu des premiers apprentissages que nous étudions dans cette thèse est

l'accès des élèves au niveau des concepts de base de l'informatique (variables, affectation, traitements alternatifs, traitements itératifs, typage des données...) : le niveau des concepts est celui où l'élève utilise des invariants conceptuels pour opérer des transferts à partir de problèmes déjà résolus lorsque se présente une tâche nouvelle, et est capable de raisonner sur les concepts indépendamment des domaines de tâches. Ce niveau suppose que l'élève dispose d'un ensemble de **S.R.T.** suffisamment différenciés et mis en relation, c'est pourquoi, les premiers apprentissages en informatique se concoivent comme la résolution de "problèmes de programmation", par laquelle l'élève se constitue un ensemble de **S.R.T.**, chacun relatif à un domaine de tâche, diversifie ces **S.R.T.**et crée des codes de passage entre eux.

Les spécificités du terrain d'expérimentation nous ont conduit à adopter le cadre général impératif (marqué par l'affectation pour le transfert des valeurs, et l'itération pour les processus répétitifs). Le fait qu'un "paradigme"[79] de programmation impose un type de structuration des données a joué également un rôle dans ce choix : l'adoption d'un langage fonctionnel aurait conduit nécessairement à la structure de liste pour la structuration des données, et nous avons montré que cette structure suppose un fonctionnement mental incluant un raisonnement sous hypothèse, nécessairement plus proche de la récurrence que la simple action répétée de l'itération. La suite de cette thèse sera donc fortement marquée par ce choix d'un cadre impératif. Pour autant, elle n'est pas dépendante du choix d'un langage dans la classe des langages impératifs: nous avons montré que, à la condition expresse de s'en tenir à des structures suffisamment générales, la démarche s'adapte sans difficulté à un large choix de langages dans la classe impérative.

De façon à prendre en compte les spécificités des représentations du sujet relatives aux objets dans la résolution d'un problème donné à un moment donné de ses acquisitions, et les interactions de ces représentations avec les représentations des traitements, nous proposons d'étudier, au sein d'un **S.R.T.**, d'une part les plans relatifs aux objets (ou données).et d'autre part les plans relatifs aux traitements (ou algorithmes).

- Chez un sujet débutant utilisant un langage impératif, les plans relatifs aux objets dans un **S.R.T.** pour une tâche donnée, font intervenir de façon liée ses représentations des éléments de base du langage (entrées-sorties, affectation, notation fonctionnelle) et des fonctions du(des) type(s) de données utilisé(s) pour les codifications, et les procédures mentales associées. Nous pensons que la constitution de plans relatifs aux objets n'est en rien immédiate, en particulier, l'utilisation, pour la résolution de problèmes de programmation de plans issus de situations familières impose un travail d'adaptation et de différentiation. En premier lieu, une conception correcte de l'affectation ne va pas de soi : les précurseurs mathématiques peuvent être absents ou inadaptés, et une présentation de l'affectation comme transformateur de prédicats n'est pas possible à ce niveau. Il est donc probable que le sujet se constitue une "signification" de l'affectation au cours de la résolution des premiers problèmes et que donc, cette signification est marquée par le type des objets sur lesquels elle porte : le sujet aurait ainsi à constituer une signification adaptée à chacun des types (numériques, chaînes, booléens..) avant de relier ces significations.
  De même la notation fonctionnelle prend, pour le débutant, des aspects bien différents d'un type à l'autre : écritures algébriques familières sur les cardinaux,

[79] Le terme de "paradigme de programmation" à été introduit par [CLOUTIER 88] (J.F. Cloutier "Apports de différents paradigmes de programmation comme autant d'outils de pensée" Actes 1er colloque francophone Didactique de l'informatique EPI) pour désigner la conjonction d'une classe de langages et des méthodes et modes de pensée associés. On parle ainsi de "paradigme impératif", "paradigme fonctionnel", de "paradigme logique". Il nous semble que cette conjonction, posée par de nombreux auteurs, constitue une évidence trop commode qu'il serait judicieux de confronter à l'analyse didactique ; en particulier, si l'on justifie le choix d'une classe de langage par l'intérêt du paradigme associé, il faudrait également discuter du type de problèmes qu'impose ce choix, en lien avec la structuration des données liée à ce "paradigme" . (Pour un exemple d'analyse faisant l'impasse sur le type de problèmes qu'implique le choix d'un langage fonctionnel, voir [ROY 90] (J.P. ROY «La programmation fonctionnelle» Informatiques n°7))

opérations moins bien connues lorsqu'elle font intervenir des ordinaux (nous avons présenté la translation, et l'homothétie), opérateurs propositionnels ayant une ressemblance trompeuse avec le langage courant dans le cas des booléens. Quant aux chaînes, jusque là les débutants ne les ont jamais considérées comme des entités calculables, et n'ont jamais utilisé les fonctions et opérateurs de comparaison sur les chaînes sous forme symbolique. Par conséquent, d'une part les éléments du **S.R.T.** relatifs à chacun des types doivent être construits, mais de plus, il n'y a véritable intégration du langage que lorsque ces éléments de représentation auront été mis en relation, de façon à ce que des transferts puissent s'opérer.

- Au sein d'un **S.R.T.** les plans relatifs aux traitements sont mis à contribution lorsque le sujet est confronté à la réalisation d'une tâche selon un schéma non séquentiel. L'acquisition de ces schémas a fait l'objet d'études épistémologiques et didactiques, dans le cadre de la programmation d'algorithmes numériques. En ce qui concerne l'itération, des difficultés ont été repérées: difficulté à concevoir des variables dont la signification est invariante, alors que leur valeur change, difficulté à les utiliser (affectation à l'intérieur de l'itération et initialisation), difficultés dans la construction de la condition d'arrêt et dans l'articulation de cette condition avec le corps d'itération, intervention de représentations erronées des capacités de contrôle du dispositif.

  Nous pensons que, si, au niveau des concepts, il est possible, et nécessaire, de raisonner sur les traitements indépendamment des données, au niveau des domaines de tâche, et par conséquent des **S.R.T.** construits lors de la résolution des premiers problèmes de programmation, la structure algorithmique et la codification des données sur lesquels ils portent sont étroitement liées. Nous voulons, par conséquent, d'une part examiner comment l'imbrication des conceptions des traitements et des conceptions des objets conduit à un point d'équilibre dans un **S.R.T.** donné, d'autre part tenter d'observer dans la durée comment se font les réajustement au sein des **S.R.T.** lorsque le raisonnement ou l'effet en retour du dispositif conduit à remettre en cause une des conceptions en jeu.

Nous avons posé comme hypothèse que la constitution par l'élève débutant d'un ensemble de **S.R.T.** lui permettant l'accès aux concepts est un processus lent qui comporte notamment comme passage obligé la construction de plans relatifs aux objets, et leur articulation avec les plans relatifs aux traitements. Les recherches réalisées dans le domaine des premiers apprentissages concernent essentiellement l'intégration de structures relatives aux traitements, et donc des problèmes portant sur des données numériques "déjà codifiées". Nous avons montré d'une part que ces problèmes font intervenir des éléments conceptuels appartenant aux mathématiques, dont les représentations devraient être prise en compte, et d'autre part qu'ils tendent à obscurcir la spécificité et les contraintes du traitement par le dispositif.

Les problèmes de programmation qui ont pour fonction la constitution des **S.R.T.**, doivent donc, à la différence de problèmes où les données sont "déjà codifiées", conduire le sujet débutant à opérer des différenciations par rapport à ses représentations des objets et traitements intuitifs. Ils doivent donc imposer une "codification" d'objets ou de données intuitifs sous forme de données informatiques et la production d'une solution comme "calcul" sur ces données codifiées. Ils doivent faire interagir les concepts informatiques de base et au contraire, éviter de faire intervenir des éléments conceptuels en dehors du champ de l'informatique. Une autre hypothèse testée dans la suite de cette thèse est donc la possibilité de construire de tels problèmes à l'aide des types de base, et de les organiser en progression. Cette construction va donc, au niveau de l'alphabétisation, dans le sens de la recontextualisation des concepts telle que nous l'avons décrite ci-dessus.

L'étude expérimentale qui suit va donc s'organiser en fonction de ces hypothèses. La partie **II** est essentiellement une série d'observations ponctuelles : en premier lieu une exploration des plans relatifs aux objets au sein des **S.R.T.** que les élèves construisent en utilisant les chaînes de caractères (chapitre **6**) mettant à profit l'étude de cette structure menée ci-dessus, puis une exploration de l'interaction au sein des **S.R.T.** entre ces conceptions et les plans relatifs à l'itération à un point de sortie (chapitre **7**): cette exploration s'appuie sur le fait que des fonctions de la structure de

chaîne (ici, la fonction `position`), utiles dans des problèmes, peuvent être construites à partir des fonctions de base, de façon itérative. Comme nous l'avons indiqué au chapitre **4**, cette première série d'observation est de type clinique. Nous la prolongeons par une étude plus quantitative au chapitre **8**.

La partie **III** est consacrée à la production et à l'expérimentation d'une progression d'exercices en fonction des choix que nous avons indiqués ci-dessus concernant les problèmes de programmation. L'expérimentation, de type clinique, nous permet d'observer dans la durée l'évolution des **S.R.T.**, et également, de commencer à caractériser une situation pédagogique en enseignement de l'informatique. La partie **IV** est consacrée à une analyse clinique, analogue à la partie **II**, mais dans le cadre des booléens.

Partie II: Représentations mentales et acquisition des données codifiées sous la forme de chaînes de caractères Etude expérimentale en classe de seconde

Chapitre 6: Etude des plans du S.R.T. relatifs aux objets

Chapitre 7: Intervention des actions complexes

Chapitre 8: Etude quantitative

Chapitre 6 Etude des plans du S.R.T. relatifs aux objets

Résolution de problèmes "directs" impliquant l'utilisation de chaînes de caractères par une classe de seconde "Option informatique". Classification des difficultés rencontrées par les élèves.

S'il est reconnu que les débutants en programmation rencontrent de nombreuses difficultés, seules les difficultés liées aux structures algorithmiques complexes, (c'est-à-dire impliquant des ruptures de séquentialité) ont fait l'objet d'études et ceci sans que le type des objets en jeu ait été analysée comme une variable. Ainsi, parmi les recherches rapportées au chapitre 2, [GREEN 77] étudie les difficultés d'interprétation de *programmes conditionnels,* [SAMURCAY 85] considère que le concept de variable, et les notions qui lui sont liées prennent leur sens avec l'*itération*, et étudie les conceptions des élèves à travers la résolution de problèmes impliquant l'usage de l'itération, [HOC 77] étudie la résolution de problèmes complexes impliquant la *décomposition en modules, l'itération,* et des choix de représentation des données, et [COHORS-FRESENBORG 87] vise l'acquisition d'une *structure itérative* sur des entiers.

Nous avons posé comme hypothèse que la constitution par l'élève débutant d'un ensemble de **S.R.T.** lui permettant l'accès aux concepts est un processus lent qui comporte notamment comme passage obligé la construction de plans relatifs aux objets, et leur articulation avec les plans relatifs aux traitements. Nous étudions particulièrement dans ce chapitre la constitution, chez des débutants à l'issue des premiers mois d'enseignement, de plans relatifs aux objets. Ces plans comprennent les représentations des éléments de base du langage (entrées-sorties, affectation, notation fonctionnelle) et des fonctions des types de données utilisés pour les codifications ainsi que les procédures mentales utilisées par le sujet pour ses anticipations. Dans le cadre d'expérimentation que nous nous sommes fixé, le premier type utilisable est le type "chaînes de caractères", qui fait lui même intervenir les types numériques ordinaux et cardinaux. Nous avons, au chapitre 5, présenté les éléments de base des langages impératifs, et les fonctions sur les chaînes de caractères qui interviennent dans les plans que nous nous proposons d'étudier. Nous en déduisons ici certains axes que nous supposons constitutifs de plans relatifs aux objets dans des **S.R.T.** influencés par le traitement des données dans le contexte familier, et donc non encore adaptés à un traitement par un dispositif informatique. S'agissant de conceptions erronées, ces éléments du **S.R.T.** créent des "*difficultés*" aux élèves, en ce sens qu'ils les conduisent à des erreurs de syntaxe, des erreurs d'exécution, ou à un résultat non conforme à leurs anticipations, qu'ils ne peuvent s'expliquer et/ou corriger.

L'observation rapportée dans ce chapitre consiste en une épreuve d'une heure sur papier, notée EPR1 dans la suite de cette thèse, passée à la fin du quatrième mois d'option informatique, et comprenant 4 exercices, puis pour certains élèves, une séance de travail sur ordinateur enregistrée au magnétophone (entretiens).

Ce chapitre se compose d'un essai de classification des *difficultés* que nous souhaitons observer (sous-chapitre 1.), puis d'une présentation de la classe, en particulier des apprentissages antérieurs des élèves (sous-chapitre 2.), d'une présentation de l'épreuve (sous-chapitre 3.) d'une analyse des réponses (sous-chapitre 4.), et des entretiens (sous-chapitre 5.), suivi de conclusions (sous-chapitre 6.). En annexe à ce chapitre (volume Annexes), on trouvera le texte de l'épreuve, une synthèse des réponses des élèves et le texte commenté de trois séries d'entretiens.

1. Classification des difficultés attendues d'élèves débutant en programmation confrontés à une tâche de résolution de problème impliquant l'emploi du type chaîne de caractères

1.1 la conception naïve du langage de programmation (Difficulté *D.Langage*).

A la suite de la lecture de [HOC 77] (chap. 2), nous avons fait l'hypothèse que, le débutant en programmation ne dispose pas d'un **Système de Représentation et de Traitement** (**S.R.T.**) qui lui permettrait de se représenter la façon dont les données sont codifiées et traitées dans l'ordinateur ; ceci conduit l'élève à des formulations dans le langage de programmation qui ont un sens pour lui, dont il pense qu'elle doivent "fonctionner", mais qui ne sont pas exécutables ou ne conduisent pas au résultat attendu. En effet, elle ne sont pas écrites pour un dispositif informatique, car l'élève ne dispose pas du **S.R.T.** nécessaire : elles sont écrites pour un dispositif virtuel correspondant à un **S.R.T.** que l'élève s'est construit, qu'il pense pertinent, et dont le langage est souvent fortement influencé par le mode de communication habituel. Nous disons que l'élève a une conception naïve du langage de programmation, en ce sens qu'il n'aperçoit pas le caractère spécifique de ce langage, comme un débutant en géométrie a une conception naïve des éléments géométriques, en ce sens qu'il leur prête les propriétés des éléments de son environnement habituel, au lieu des les considérer comme des objets formalisés définis par un ensemble de propriétés[80]. Cette conception naïve du langage conduit l'élève à des formulations comportant des erreurs de syntaxe, et/ou il obtient à l'exécution un résultat non conforme à son anticipation. Dans le premier cas, ces formulations ne doivent pas être confondues avec de simples défauts de syntaxes ; elles persistent en effet malgré le message d'erreur du compilateur, et souvent aussi après que l'enseignant ait rappelé la syntaxe correcte.

Nous avons présenté au chapitre précédent les structures fondamentales des langages "impératifs" que les élèves observés utilisent. Nous avons vu que, dans ces langages, et si l'on n'envisage pas d'actions complexes, le programmeur débutant dispose seulement des actions d'affectation et d'entrée/sortie, et utilise les fonctions prédéfinies. Nous devons donc chercher les difficultés concernant le langage, dans ces deux directions.

Les actions : dans les exercices que nous proposerons, l'instruction d'entrée étant située en début de programme, ne modifie pas les rapports entre les variables. L'affectation est donc la seule action ayant une influence sur l'état des variables du programme. Nous avons montré qu'il n'est pas possible que l'enseignant présente aux débutants l'affectation comme transformateur d'assertions, et qu'une présentation de l'affectation à partir d'un dispositif de bas niveau (mémoire comme ensemble de tiroirs) donne une idée trop locale des effets de l'affectation. Nous avons mis en évidence des précurseurs en mathématiques (par exemple, dans un problème, on substituera une variable à une expression complexe pour se ramener à un problème plus simple). Mais ces précurseurs peuvent ne pas être présents chez tous les élèves, et dans l'enseignement qu'on reçu les élèves, ils concernent seulement des variables s'apparentant aux cardinaux, alors que nous voulons observer la résolution de problèmes sur des données de type chaîne ou ordinaux. Quelque soit la présentation

---

[80] [DAGDILELIS 91] décrit de la façon suivante ce qu'il appelle «*l'hypothèse antropomrophique*», suite à [PEA 84]: «*Ainsi les élèves ont tendance à attribuer aux instructions d'un programme une sémantique beaucoup plus large que la sémantique réelle. Ils ne croient pas vraiment à l'existence d'une intelligence humaine "cachée" dans la machine. Mais souvent ils se comportent comme s'il y en avait une.*». Pour nous, les débutants ne peuvent pas accéder d'emblée à la compréhension des spécificités d'un langage de programmation. En effet les notations formelles qu'ils ont pu rencontrer (par exemple l'algèbre) ne sont pas des langages de commande. C'est pourquoi ils prêtent au langage de programmation les propriétés du langage dans la communication habituelle qui est bien entendu anthropomorphique, qu'elle s'adresse à des humains, à des animaux et même à certains dispositifs (par exemple l'ordinateur Macintosh de Apple affiche un large sourire lors d'un amorçage réussi).

adoptée par l'enseignant, il est donc à prévoir qu'une conception adaptée de l'affectation sera absente chez de nombreux élèves. En particulier, l'affectation modifie l'état interne du dispositif, mais elle ne produit rien de visible extérieurement ; par contre, les actions de sortie constituent des effets visibles extérieurement, alors qu'elles sont indifférentes du point de vue de l'état des variables du programme. Les élèves vont ils comprendre le rôle de l'affectation dans l'évolution de l'état du dispositif ? Ou vont-ils penser que d'autres éléments du langage tiennent ce rôle ? Les situations que nous nous proposons d'observer dans ce chapitre ne mettent pas en jeu l'itération, et donc l'affectation est sollicitée seulement à un niveau assez élémentaire ; nous voulons observer si, cependant, des difficultés se présentent dès ce niveau.

Les fonctions : nous avons montré au chapitre précédent que l'emploi des fonctions pré-définies du langage implique la construction d'écritures fonctionnelles qui peuvent être nouvelles pour les élèves ; les calculs successifs sur les objets impliquent la composition des fonctions ; cette composition peut se faire à l'aide d'écritures fonctionnelles imbriquées, ou par l'affectation à des variables des résultats intermédiaires. Nous avons noté que, quand on se limite aux types numériques, les expressions imbriquées peuvent ne pas être comprises par les élèves comme fonctionnelles, car les fonctions sont en fait les opérations habituelles, dont les élèves ont intégré les règles d'emploi dans le domaine mathématique, au moment de l'acquisition du calcul algébrique. Un calcul complexe sur un type numérique peut être considéré par les élèves comme une expression parenthésée avec priorités, et non comme une composée de fonctions. Par contre, avec les chaînes de caractères, sont introduites des fonctions telles que les élèves en ont peu rencontrées : fonctions de plusieurs variables, dont les arguments ne sont pas nécessairement numériques, et dont la syntaxe est réellement fonctionnelle. Comme pour l'affectation, nous avons montré que les précurseurs en mathématiques peuvent ne pas être présents, et que la conception du traitement d'une écriture fonctionnelle par le dispositif n'est pas immédiate pour le débutant.

Les problèmes dont nous nous proposons d'observer la résolution dans ce chapitre consistent à construire des fonctions sur des chaînes comme composées de fonctions prédéfinies ; ils mettent donc en jeu la syntaxe fonctionnelle et la composition de fonctions, dans un cadre qui ne réduit pas à celui des expressions algébriques rencontrées par les élèves en mathématiques. Si nos hypothèses sont exactes, nous devons nous attendre à observer dans la résolution de ces problèmes, des difficultés (que nous noterons de façon abrégée dans la suite *D.Langage*) conduisant l'élève à des écritures erronées, mais qui ont un sens pour lui, témoignant d'une incompréhension du rôle de ces éléments du langage que sont l'affectation et la notation fonctionnelle.

1.2 Les difficultés propres à la structure de chaîne (Difficulté *D.Sem.Cha.*)

A l'intérieur d'un langage général correspondant à un dispositif, un type constitue une catégorie particulière, avec ses régles syntaxiques de formation des expressions et les propriétés qui définissent sa sémantique. Comme les régles et les propriétés générales du langage, elles sont rencontrées et intégrées progressivement par le sujet, de façon à former un **S.R.T.** particulier à ce type, au sein du **S.R.T.** général. Avec des élèves débutants utilisant les chaînes de caractères, on doit s'attendre à trouver des formulations qui marquent l'absence d'un **S.R.T.** adéquat. On peut imaginer qu'un sujet ayant intégré les contraintes générales d'un langage comportant déjà certains types, rencontre des difficultés avec un type nouveau : il doit installer un **S.R.T.** nouveau, au sein d'un **S.R.T.** général adéquat. Tel n'est pas le cas des élèves débutants : à travers les premiers types qu'ils rencontrent, ils doivent à la fois construire le **S.R.T.** global, et des **S.R.T.** particuliers correspondant aux premiers types qu'ils rencontrent : ainsi l'affectation comme structure générale se comprend au début à travers l'affectation à des variables de type particulier. On doit donc s'attendre à des difficultés (que nous noterons de façon abrégée dans la suite *D.Sem.Cha.*) s'analysant comme résultant d'une non-intégration des régles et des propriétés du type "chaînes de caractères", mais également fortement reliées aux difficultés résultant d'une conception naïve du langage et décrites ci-dessus (*D.Langage*).

1.3 La conception des ordinaux, et la coordination ordinaux-cardinaux (Difficulté *D.Ord.Card.*)

Nous avons souligné (chapitre précédent), que les ordinaux sont l'objet d'apprentissages au tout début de l'enseignement des mathématiques (première année d'enseignement obligatoire), et que ces apprentissages visent la coordination entre les deux aspects ordinal et cardinal du nombre. L'aspect cardinal fait ensuite seul l'objet d'apprentissages, la structure cardinale étant étendue du discret au continu, à travers les rationnels puis les réels. De même, la notation algébrique (c'est à dire l'introduction de lettres dans les calculs, représentatives de quantités) concerne seulement des quantités apparentées aux cardinaux (entiers, réels, vecteurs...).

Nous avons souligné que les ordinaux interviennent comme second argument de la fonction sous-chaîne, et comme résultat de la fonction longueur (si l'on considère ce résultat comme la position du dernier caractère de la chaîne argument). Les cardinaux interviennent comme troisième argument de la fonction sous-chaîne, et comme résultat de la fonction longueur (si l'on considère ce résultat comme le nombre de caractères de la chaîne argument). Des opérations comme la translation (ajout d'un cardinal fixe à un ordinal variable), ou l'homothétie (`n` étant un cardinal, l'homothétie de rapport `n` fait passer d'un ordinal `i` à l'ordinal `j`, rang du premier caractère du `i`ème groupement de `n` caractères dans une chaîne) se révèlent utiles pour le traitement des chaînes, et sollicitent la coordination entre les ordinaux et les cardinaux et l'emploi de la notation algébrique pour les ordinaux. Nous avons fait le choix ici observer les difficultés de coordination ordinaux-cardinaux à partir de problèmes mettant en jeu la fonction longueur et l'homothétie.

1.4 Le typage des objets (Difficulté *D.Type*)

Dans les problèmes dont nous projetons d'observer la résolution, l'élève doit concrétiser la codification des données par le choix d'un type prédéfini du langage (ce choix est réduit au début aux chaînes et aux nombres), et la construction de fonctions adaptées à la résolution du problème à l'aide des fonctions du type choisi. Pour l'élève, le typage des objets entraîne d'une part un choix de représentation, et d'autre part des contraintes syntaxiques. Le choix du type prédéfini utilisé pour construire la représentation d'un objet donné doit être guidé par la commodité que ce type présente pour implémenter les fonctions nécessaires à la résolution du problème, et suppose donc une abstraction au moins implicite. Les contraintes syntaxiques consistent dans la nécessité de déclarer le type des variables, et de contrôler que, dans l'écriture du programme, les arguments des fonctions sont cohérents avec la définition du type.

On peut considérer que les élèves ont rencontré antérieurement, en mathématiques, des notations analogues au typage des objets : en géométrie, par exemple, on précise par une typographie particulière si une lettre désigne un vecteur, un point, un nombre ou une fonction, et on peut repérer des expressions qui ont un sens, en fonction de la nature des opérandes : l'addition de deux vecteurs a un sens, également l'addition d'un point et d'un vecteur, mais pas l'addition d'un nombre et d'un vecteur, ni la multiplication de deux points. A un niveau qui dépasse celui des élèves auxquels nous nous intéressons, le typage n'est pas figé : des fonctions peuvent par exemple, être considérées comme des vecteurs d'un espace (c'est le cas de l'ensemble des polynômes qui constitue un espace vectoriel). Par contre, pour les élèves du secondaire, en tant que modélisation des relations spatiales, la géométrie établit une correspondance "naturelle" entre les objets sensibles et les objets géométriques. Il arrive que des éléments soient considérés sous deux aspects : depuis l'école élémentaire, les élèves manipulent par exemple les nombres à la fois comme entité numérique, et comme chaîne de chiffres, passant d'un aspect à l'autre à l'occasion des mécanismes opératoires. Mais, sauf cas particulier de changement de base de numération, les élèves n'ont pas besoin de considérer les nombres et les chaînes de chiffres comme des entités distinctes.

Nous considérerons donc que la nécessité (syntaxique) d'une déclaration de type peut s'inscrire dans une continuité avec les apprentissages antérieurs. Par contre, la nécessité de choisir le type de base utilisé pour la représentation en considérant une

abstraction au moins implicite des données, et les propriétés respectives des types de base, et non selon l'apparence externe des données manipulées, ainsi que la nécessité d'employer explicitement des fonctions de conversion de type, sont en rupture avec les apprentissages mathématiques antérieurs. Nous proposerons donc un exercice très simple impliquant ce choix.

2. Apprentissages antérieurs des élèves concernés par l'épreuve

Nous avons fait passer l'épreuve EPR1 courant janvier 86, après un premier trimestre assez long, mais pendant plusieurs semaines les cours n'avaient pas eu lieu du fait du mouvement lycéen contre la réforme MONORY. Les élèves ont eu précédemment une trentaine d'heures de cours et de travaux dirigés. Le cours a introduit les notions suivantes : d'une part les "objets", avec distinction des types : nombres, chaînes de caractères, booléens, d'autre part les "instructions" : instructions d'affichage, de lecture ; instructions sur les sous-programmes (appel, retour) ; instruction d'affectation...

Les problèmes résolus par les élèves en travaux dirigés antérieurement à l'épreuve ont porté essentiellement sur la gestion de l'écran texte, sur la séquentialité et sur l'alternative. Nous les résumons ci-dessous, et nous discutons la façon dont ces problèmes font intervenir les difficultés que nous avons présenté ci-dessus :

Problème 1. affichage à l'écran de la phrase "`vive l'informatique`", encadrée d'étoiles.

Problème 2. affichage à l'écran d'un phrase entrée par l'utilisateur du programme, et encadrée d'étoiles.

Problème 3. affichage à l'écran d'un "menu" de restaurant, avec choix de la part de l'utilisateur, et affichage du prix total.

Problème 4. Quelques élèves ont, en travaux pratiques sur ordinateur résolu un exercice de conjugaison des verbes du troisième groupe, à l'aide des fonctions sur les chaînes de caractères.

Dans le problème 2. la difficulté principale consiste à calculer la chaîne à afficher en première ligne (c'est une chaîne composée d'étoiles en nombre égal à la longueur de la chaîne à afficher plus 2. Le professeur a proposé l'utilisation d'une variable `ETOILE$` initialisée à "**********" ; la solution consiste à calculer à l'aide de la fonction `LEN` (longueur) le nombre d'étoiles nécessaires pour encadrer la phrase, et à calculer à partir de `ETOILE$` la sous-chaîne correspondante. Les fonctions sur les chaînes de caractères, introduites en cours, ont donc été utilisées à cette occasion.

La tâche sous-jacente aux problèmes posés, ainsi que les exigences du professeur en matière d'écriture du programme conduisent à ce que **la programmation consiste à obtenir des effets à l'écran**. Nous avons dit plus haut que les instructions de sortie n'interviennent pas dans la sémantique du programme ; ce n'est pas entièrement vrai dans ces problèmes, puisque l'aspect final de l'écran constitue le résultat. Dans ce type de programmation, il faut en fait considérer l'écran comme une variable, les instructions d'affichage transformant les assertions intervenant sur cette variable. Le traitement de cette variable fait intervenir des éléments complexes : prise en compte de l'organisation de l'écran en ligne, défilement de l'écran, position du curseur...

Le choix de ce type de programmation est assez répandu en initiation ; de nombreux apprentissages en LOGO, par exemple, sont centrés sur le graphisme à l'écran. Les hypothèses didactiques sous-tendues par ce choix ne nous paraissent pas avoir été élucidées : dans LOGO, comme nous l'avons vu au chapitre 3, ce choix est guidé par la volonté de ne pas réduire la problématique à celle de l'apprentissage de la programmation : [PAPERT 80] situe ses objectifs dans le domaine d'apprentissages "transversaux" (apprendre à penser) ou mathématiques (Géométrie), mais ne donne pas de précision sur les conditions de mise en oeuvre de ce programme. C'est pourquoi,

nous préférons observer la résolution de problèmes où il y a clairement une donnée et un résultat calculé, l'instruction de sortie servant seulement à communiquer ce résultat à l'utilisateur du programme.

Une autre caractéristique de l'enseignement reçu par les élèves est que, en dehors du problème 4 (conjugaisons), résolu seulement par certains élèves, **les chaînes de caractères n'ont pas été utilisées pour des données où la structure de chaîne soit réellement sollicitée** : le problème 2 (affichage d'une chaîne de longueur variable, encadrée d'étoiles) consiste en un calcul sur une chaîne répétition d'un même caractère; la question de la longueur (cardinale) de la sous-chaîne à extraire se pose donc, mais pas celle du rang (ordinal) où commence la sous-chaîne à extraire. Dans l'exercice de conjugaison, lors du calcul du radical, la position de début est 1, et donc la longueur de la sous-chaîne à extraire se confond avec la position du dernier caractère extrait.

Les difficultés que nous avons exposées au sous-chapitre précédent sont donc peu sollicitées. De plus, les problèmes que nous voulons proposer aux élèves sont différents de ceux que les élèves ont résolu en travail dirigé, en ce sens qu'ils impliquent qu'il y ait clairement des données et des résultats, et s'éloignent donc de problèmes où la tâche à réaliser est constituée d'effets sur l'écran. Il ne s'agit pas ici d'évaluer un enseignement, mais d'étudier les difficultés des élèves selon la grille d'analyse exposée au sous-chapitre précédent ; nous nous contenterons donc de considérer que les élèves connaissent les fonctions sur les chaînes, et les principes généraux de l'écriture d'un programme, mais n'ont pas été particulièrement préparés à l'épreuve.

3. Présentation de l'épreuve EPR1

L'épreuve sur papier s'est déroulée sur une heure, onze élèves étant présents. Lors des séances suivantes, nous avons travaillé avec des groupes d'élèves en présence de l'ordinateur à partir de leurs réponses à l'épreuve sur papier ; ces entretiens ont été enregistrés, et les différentes versions du programme ont été prises en note. On trouvera ci-dessous une présentation question par question du texte de l'épreuve sur papier, puis une analyse des réponses des élèves, et des entretiens. En annexe à ce chapitre, on trouvera le texte de l'épreuve (Annexe 7.1), suivi des réponses des élèves(Annexe 7.2), et du dépouillement des entretiens (Annexe 7.3 à 7.5).

Le texte comporte un rappel sur la fonction sous-chaîne (`MID$` en BASIC), renforcé par un exemple :

Rappel : MID$(X$,a,b) où X$ est une variable chaîne, a et b deux nombres, est la chaîne constituée de b caractères pris dans X$, à partir du a-ième caractère.

Exemple : Ce programme affiche "NJOU"

```
10 LET X$="BONJOUR"
20 LET Y$=MID$(X$,3,4)
30 PRINT Y$
40 END
```

Une première question (question 1) vise à donner aux élèves un cadre syntaxique pour l'expression de leurs réponses aux questions suivantes ; elle permet également de vérifier que les élèves ne font pas d'erreurs dès le cas élémentaire de la recherche d'une sous-chaîne de longueur 1.

On suppose qu'on a écrit le programme suivant :

```
10 INPUT X$ : REM entrée d'une chaîne au clavier
20 LET A$=MID$(X$,1,1)
30 LET L=LEN(X$)
40 LET B$=MID$(X$,L,1)
50 PRINT A$;B$
60 END
```

Indique ce que l'ordinateur affiche à la ligne 50, 1.a dans le cas où l'utilisateur a entré la chaîne "BONJOUR" à la ligne 10 .

1.b dans le cas où l'utilisateur a entré la chaîne "B" à la ligne 10 .

3.1 Les problèmes ONJOURB et RONJOUB

Dans les deux exercices suivants, on a une chaîne en entrée, on veut une chaîne en sortie, modifiée selon une loi donnée. Nous les groupons, car les difficultés attendues sont très proches ; en fait nous attendons surtout que le problème **RONJOUB** confirme des difficultés rencontrées avec le problème **ONJOURB**.

question 2 (problème **ONJOURB**)

Ecris un programme pour que l'utilisateur, ayant entré une chaîne au clavier, l'ordinateur affiche la chaîne en passant la première lettre à la fin.

Exemple : si l'utilisateur entre "BONJOUR", l'ordinateur affichera "ONJOURB".

question 4 (problème **RONJOUB**)

Ecris un programme pour que l'utilisateur ayant entré une chaîne au clavier, l'ordinateur affiche le mot en intervertissant la première et la dernière lettre.

Exemple : si l'utilisateur entre "SALUT", l'ordinateur affichera "TALUS".

La solution peut être obtenue par une composition déterminée des fonctions sous-chaîne, longueur, et concaténation :

par exemple, pour le problème **ONJOURB** en BASIC :

```
LET resultat= MID$(chaîne,2,LENGTH(chaîne)-1) +
              MID$(chaîne,1,1)
```

et pour le problème **RONJOUB**:

```
LET resultat:= MID$(chaîne,LENGTH(chaîne),1) +
               MID$(chaîne,2,LENGTH(chaîne)-2) +
               MID$(chaîne,1,1)
```

L'affectation des résultats intermédiaires à des variables permet éventuellement de morceler chaque formule :

pour le problème **ONJOURB**

```
LET L= LENGTH(chaîne)
LET corps=MID$(chaîne,2,L-1)
LET tête= MID$(chaîne,1,1)
LET résultat=corps + tête
PRINT résultat
```

et pour le problème **RONJOUB**:

```
L= LENGTH(chaîne)
LET corps=MID$(chaîne,2,L-1)
LET tête= MID$(chaîne,1,1)
LET queue= MID$(chaîne,LENGTH(chaîne),1)
LET résultat= queue + corps + tête
PRINT résultat
```

On attend de cet exercice des indications sur les difficultés des élèves concernant les éléments généraux du langage (*D.Langage*), et celles qui concernent spécifiquement les chaînes (*D.Sem.Cha.*).

Les solutions présentées ci-dessus sont conformes à une conception de la programmation qui fait passer des données au résultat ; il est cependant possible aux élèves de se livrer à des affichages successifs, dans l'esprit des exercices qu'ils ont résolu en travaux dirigés, évitant ainsi le calcul du résultat par concaténation, et les difficultés qui en résultent. Par exemple pour le problème **ONJOURB**

```
LET L= LENGTH(chaîne)
LET corps=MID$(chaîne,2,L-1)
PRINT corps; (; pour afficher-sans-passer-à-la-ligne)
LET tête= MID$(chaîne,1,1)
PRINT tête
```

3.2 Le problème NOTES (question 3)

On a en entrée une donnée numérique (ordinale) ; on veut en sortie une sous-chaîne extraite (de la chaîne constante du problème). Il s'agit de la construction à l'aide de la structure "chaînes de caractères" d'une fonction d'accès à des données organisées dans une structure analogue à celle de tableau de chaînes de caractères.

Complète le programme ci-dessous (... ligne 30) pour que, l'utilisateur ayant entré un nombre entre 1 et 7, l'ordinateur affiche la note correspondante de la gamme (SO au lieu de SOL):

```
10 LET NOTE$="DO*RE*MI*FA*SO*LA*SI"
20 INPUT I
30 PRINT MID$(NOTE$,...,2)
```

La tâche consiste à construire la fonction qui à l'ordinal donné en entrée fait correspondre l'ordinal marquant le début de la chaîne à extraire. Cette fonction est ce que nous avons appelé dans l'étude de la structure ordinale une "homothétie". a_0 étant le plus petit ordinal, et a étant un ordinal donné, la fonction lui associe $a_0 + 3(a-a_0)$. Le résultat peut donc être calculé comme `MID$(NOTE$, 1 + 3*(I-1),2)`, ou dans une autre forme, comme `MID$(NOTE$, 3*I-2,2)` . S'il ne reconnaissent pas une "homothétie" ordinale, les élèves peuvent utiliser leurs connaissances en mathématiques, concernant les fonctions "affines", c'est à dire pour eux, les fonctions du type `f(x) = a x + b`. En effet, ils ont dû rencontrer ce type de fonction dans des situations mettant en jeu des cardinaux ou dérivés : ce sont toutes les situations où l'on a proportionnalité entre l'accroissement de deux grandeurs, par exemple la longueur d'une tige de fer et la température, dans une étude de la dilatation des métaux. Une façon courante de rechercher cette fonction consiste à repérer le coefficient de proportionnalité `a` (ici ce serait 3) dans les données du problème, puis à rechercher `b` sur une valeur particulière, par exemple la valeur initiale.

L'exercice sollicite donc les difficultés de coordination ordinaux-cardinaux (*D.Ord.Card.*)

3.2 Le problème CHIFFRE (Question 5)

Ecris un programme pour que, l'utilisateur ayant entré un nombre au clavier, l'ordinateur affiche le chiffre des unités.

Les élèves ne disposent pas des fonctions numériques (partie entière ou modulo) qui leur permettraient de résoudre le problème de façon numérique. Ils ont la possibilité pour résoudre simplement cet exercice d'utiliser les fonctions `sous-chaîne`, et `longueur` , d'une manière qui est d'ailleurs indiquée à la question 1. Pour cela, ils doivent accepter d'utiliser des fonctions sur les chaînes de caractères pour un problème d'apparence numérique, c'est à dire d'une façon plus générale, de faire le choix d'un type de donnée, non en fonction de l'apparence externe des données manipulées, mais en fonction des opérations sur ces données que la résolution du problème nécessite. Nous avons appelé *D.type* la difficulté à faire ce choix.

4. Exploitation des réponses des élèves

Les réponses des élèves, question par question peuvent être consultées dans le volume Annexes pages 24 à 27.

4.1 Les problèmes ONJOURB et RONJOUB

La tâche est bien comprise pour le problème **ONJOURB** . Trois élèves (Stéphanie G., Alexandra J., Florence P.) n'utilisent pas de variables auxiliaires. Leurs réponses paraissent assez proche de la solution attendue (variante utilisant la composition des applications sans variables auxiliaires), mais comportent des erreurs qu'on peut interpréter en première analyse comme des défauts de syntaxe. Un travail sur ordinateur (entretien) mené avec Stéphanie G. et une autre élève permettra de pousser plus loin l'analyse.

Chez deux élèves (Florent R., Bernard G.), le calcul dans le problème **ONJOURB** est décomposé à l'aide de variables auxiliaires, mais la réponse présente des erreurs liées à la difficulté *D.Langage* (concernant l'affectation) ou *D.Ord.Card.* (coordination ordinaux-cardinaux).

Pour les cinq autres élèves (soit un peu moins de la moitié des élèves), le calcul est décomposé, les parties sont correctement isolées et restituées dans l'ordre attendu, mais les élèves n'emploient pas la concaténation.

Le problème **RONJOUB** donne lieu à une interprétation fausse de l'énoncé chez un élève (Bernard G. se donne pour tâche de "retourner" le mot donné, c'est-à-dire de résoudre un problème RUOJNOB) ; ne disposant pas des compétences lui permettant de résoudre le problème qu'il s'est posé, il produit un programme non-exécutable.

Le problème **RONJOUB** n'est pas traité par 4 élèves, parmi lesquels trois avaient répondu de façon erronée au problème **ONJOURB**. La quatrième élève ayant produit une réponse erronée au problème **ONJOURB** (Stéphanie G.) décompose la tâche en deux sous-tâches, présentant des erreurs similaires à celles relevées au problème **ONJOURB**.

Une élève ayant répondu correctement au problème **ONJOURB** (Caroline C.) décompose elle aussi le résultat en deux résultats intermédiaires, mais il y a des erreurs liées à *D.Sem.Cha.* : incompréhension du rôle des arguments de la fonction sous-chaîne ; affichage des éléments de la chaîne donnée dans l'ordre initial, et non dans l'ordre pertinent pour le résultat.

Parmi les 4 réponses restantes, 3 présentent des erreurs liées à *D.Ord.Card* ; en particulier pour deux élèves, dans une chaîne de longueur `L`, la position du dernier caractère est compris comme `L-1`, au lieu de `L`. Une réponse est correcte.

	RONJOUB					
ONJOURB	D.Langage	D.Sem.Cha.	D.ord.card	correct	non fait	total
D.Langage	1	1	0	0	3	5
D.Sem.Cha	1	0	0	0	0	1
correct	0	0	3	1	1	5
total	2	1	3	1	1	11

Le tableau ci-dessus résume les difficultés rencontrées par les élèves dans les problèmes **ONJOURB**) et **RONJOUB**). Les élèves qui rencontrent les difficultés *D.Langage* et *D.Sem.Cha.* dans le problème **ONJOURB** les rencontrent à nouveau dans le problème **RONJOUB**, ou ne répondent pas. Ceux qui réussissent à **ONJOURB** rencontrent des difficultés nouvelles dans **RONJOUB**, essentiellement *D.Ord.Card.* Un seul élève réussit aux deux questions. Les difficultés *D.Langage* et *D.Sem.Cha.* paraissent donc peu différenciées. Par contre, des élèves qui paraissent avoir bien assimilé les spécificités du langage, et la signification des fonctions sur les chaînes peuvent échouer à cause de difficultés concernant les ordinaux ; ils utilisent tous l'affectation des résultats intermédiaires à une (des) variable(s) auxiliaire(s). Chez ces élèves, il n'apparaît pas de difficultés concernant la concaténation, car ils n'emploient pas cette opération ; ils emploient l'affichage sans passer à la ligne pour restituer dans l'ordre attendu les résultats partiels qu'ils ont calculé. Il est a posteriori, regrettable que le texte de l'épreuve n'ait pas incité davantage les élèves à calculer le résultat avant de l'afficher.

4.2 Le problème NOTES (Question 3)

La tâche est la construction d'une fonction "homothétie" de rapport 3 dans les ordinaux. Seuls 5 élèves sur 11 mènent à bien cette tâche. L'observation menée pendant la passation montre qu'ils ont procédé par tâtonnement à partir du tableau de valeurs de la fonction ; le caractère affine de la fonction n'est apparu comme découlant ni de la définition de la fonction, ni de son tableau de valeur ; la fonction cube (laissant 1 invariant, et faisant intervenir le nombre 3 comme opérateur) a été essayée par

plusieurs élèves et n'a été rejetée qu'après vérification expérimentale.

Nos hypothèses sur les difficultés spécifiques aux ordinaux se confirment ici ; qu'ils réussissent ou qu'ils échouent, les élèves ne transfèrent pas dans le domaine des ordinaux leurs connaissances concernant les fonctions affines vues en mathématiques sur les cardinaux ou dérivés.

4.3 Le problème CHIFFRE (Question 5)

Seuls trois élèves répondent à cette question ; les élèves disposaient d'une heure pour l'ensemble de l'épreuve, ce qui semble suffisant, même s'ils ont passé beaucoup de temps sur 3 et 4 ; un effet de lassitude a pu se produire pour ce dernier exercice ; on peut cependant penser que la non-réponse de 8 élèves sur 11 s'explique en partie par l'absence d'une solution évidente et donc la difficulté notée *D.type* à utiliser la structure "chaîne de caractères" pour un problème apparemment numérique.

Les trois élèves qui répondent utilisent les fonctions "`longueur`" et "`sous-chaîne`" ; l'écriture est correcte pour l'un (Bernard G.), et comporte dans les 2 autres cas (Armelle D., Karine C.) l'erreur (de type *D.Ord.Card.*) sur la position du dernier caractère de la chaîne, déjà constatée à l'exercice 4 (problème **RONJOUB**).

Par contre, ces 3 élèves font porter les fonctions sur des variables numériques (identificateurs non terminés par un "$", en BASIC) ce qui entraînerait une erreur de type : argument numérique à la place d'un argument chaîne. Nous rechercherons dans les entretiens rapportés ci-dessous s'il s'agit d'un simple défaut de syntaxe, ou si cette erreur est révélatrice d'une difficulté à manipuler les types (*D.type*) qui conduit les élèves à rechercher une représentation des données conforme à l'aspect externe des quantités codifiées, et non aux fonctions que l'on souhaite leur appliquer.

4.4 Récapitulation par élève

Les 5 premières colonnes correspondent au succès ou aux difficultés repérées dans chaque exercice ; dans la sixième colonne, on a tenté de repérer une difficulté dominante, mais cela n'a été possible que pour 4 élèves ; la dernière colonne donne un indice de réussite (nombre d'exercices correctement résolus sur 5).

	question 1	**ONJOURB** question 2	**NOTES** quest.3	**RONJOUB** question 4	**CHIFFRE** question 5	difficulté dominante	I R
Caroline C.	correct	D.Langage	non fait	D.Sem.Cha.	non fait		1
Bernard G.	D.Sem.Cha	D.Sem.Cha	non fait	non fait	D.type	D.Sem.Cha	0
Florence P.	D.Sem.Cha	D.Langage	non fait	non fait	non fait		0
Stéphanie G	correct	D.Langage	non fait	D.Langage	non fait	D.Langage	1
Pierre P.	correct	correct	correct	non fait	non fait		3
Florent R.	correct	D.Langage	non fait	non fait	non fait		1
Karine C.	correct	correct	correct	D.Ord.Card	D.Ord.Card	D.Ord.Card	3
					D.type		
Armelle D.	correct	correct	correct	D.Ord.Card	D.Ord.Card	D.Ord.Card	3
					D.type		
Pierre E.	correct	correct	non fait	D.Ord.Card	non fait		2
Frédéric P.	correct	correct	correct	correct	non fait		4
Alexandra J	D.Sem.Cha	D.Langage	non fait	non fait	non fait		0

4.5 Conclusions

- Les difficultés *D.Sem.Cha.* et *D.Langage* sont variées et semblent liées chez les élèves:
  - dans la question 1, mauvaise interprétation du programme (3 élèves),
  - difficulté à articuler deux "extractions" de sous-chaîne (Stéphanie G. dans le problème RONJOUB),
  - interprétation du deuxième argument comme la position du caractère dans le

résultat, et non la position au moment du calcul (Bernard G.),
- influence des formulations en langage naturelle sur les expressions utilisant les fonctions (Florent R.),
- écritures syntaxiquement correctes, mais n'aboutissant pas au résultat (Caroline C.).

- Les erreurs provenant de difficultés *D.Langage* sont présentes chez les élèves qui n'utilisent pas l'affectation des résultats intermédiaires comme méthode de décomposition du problème. Les élèves qui utilisent systématiquement la décomposition à l'aide de l'affectation à des variables intermédiaires semblent mieux utiliser le langage. Cette constatation nous conduit à deux hypothèses.
  - Hypothèse 1 : leurs écritures étant moins complexes, ces élèves les dominent mieux ; dans cette hypothèse, l'emploi de l'affectation pour les résultats intermédiaires conduirait à une meilleure intégration du langage.
  - Hypothèse 2 : ayant intégré le rôle de l'affectation, et la décomposition du problème en sous-tâches, ils sont capables de faire cette décomposition ; dans cette hypothèse, la production d'écritures moins complexes serait une conséquence de la meilleure intégration du langage.

  Ces hypothèses sont-elles incompatibles? Pas nécessairement, si l'on considère que dans la genèse des acquisitions la production d'écritures plus facilement dominées et la compréhension des spécificités du langage peuvent progresser de pair. Nous étudierons aux chapitres **9** et **10** comment s'opère cette progression en observant sur une dizaine de séances quatre élèves particulièrement en difficulté.
  L'épreuve sur papier ne permet pas de connaître la signification pour l'élève des écritures erronées (qu'elles soient de nature *D.Langage* ou *D.Sem.Cha.*) ; en particulier on ne peut savoir si ces erreurs sont liées à une incompréhension globale du langage, ou s'il s'agit de simples erreurs d'anticipation, le "feed back" donné par une exécution sur ordinateur permettant dans ce cas d'obtenir un programme correct par corrections locales. C'est pourquoi nous ferons porter l'observation sur ce point dans les entretiens sur ordinateur.

- Les difficultés *D.Ord.Card* sont nombreuses dès qu'il s'agit de calculer des ordinaux ou des cardinaux :
  - rang du dernier élément, et longueur de sous-chaînes (4 élèves).
  - impossibilité de construire la fonction (ordinale) dans le problème **NOTES** pour 6 élèves.

  Ceci confirme nos hypothèses quant à la non-disponibilité des ordinaux et aux difficultés de coordination ordinaux-cardinaux chez les élèves de seconde. Mais de même que pour les difficultés *D.Langage* et *D.Sem.Cha.*, nous devons déterminer si les élèves ont, malgré les difficultés constatées dans l'épreuve sur papier, une connaissance suffisante des ordinaux pour corriger leurs erreurs (en particulier celle concernant la position du dernier élément) suite l'exécution sur ordinateur.

- La difficulté *D.type* : seuls 3 élèves sur 11 résolvent le problème 5, et ils utilisent les fonctions de traitement de chaîne sur des données numériques. On peut penser que les 8 élèves qui ne répondent pas hésitent à utiliser des fonctions de traitement de chaînes pour une donnée qui leur apparaît comme numérique, ou ne concoivent pas que les fonctions sur les chaînes apportent ici une solution : la "nature" intuitive des données serait un obstacle à leur abstraction. La réponse des 3 autres montre que cette abstraction est réalisée, puisqu'ils assimilent les données numériques à des chaînes. Le type numérique en BASIC n'imposant pas de déclaration particulière, l'absence de déclaration des données de type chaîne peut soit s'expliquer par un "oubli" de la déclaration de type, et donc s'analyser comme un défaut de syntaxe facilement corrigé à la suite du message d'erreur du compilateur ou de l'interpréteur, soit constituer le choix conscient du type numérique, malgré l'intention d'employer les fonctions de traitement de chaîne. Dans ce cas, l'absence de déclaration du type chaîne résulterait bien du choix par les élèves du type selon la nature intuitive des données, et non selon le jeu de fonctions que l'on souhaite appliquer aux données suite à une abstraction implicite. Cette difficulté sera donc

analysée plus précisément dans un entretien.

5. Les entretiens

Comme nous l'avons annoncé ci-dessus, nous avons travaillé avec des groupes d'élèves en présence de l'ordinateur à partir de leurs réponses à l'épreuve sur papier. Nous avons travaillé avec un groupe de deux élèves rencontrant des difficultés *D.Langage* et *D.Sem.Cha.*, en deux phases, l'une portant sur le problème **ONJOURB**, l'autre sur le problème **RONJOUB**. Nous avons travaillé ensuite avec un autre groupe de deux élèves qui, dans l'épreuve sur papier, nous avaient semblé maîtriser suffisamment les structures générales du langage mais rencontrer les difficultés *D.Ord.Card.* et *D.Type* que nous souhaitions analyser davantage. Puis nous avons eu un entretien avec un élève absent à l'épreuve écrite, mais réussissant en général bien aux exercices proposés par le professeur ; nous voulions, dans cet entretien repérer les capacités particulières que met en oeuvre un élève qui réussit, en particulier les écritures fonctionnelles composées. Le dépouillement commenté de l'enregistrement au magnétophone de ces 3 séries d'entretiens se trouve dans le volume Annexes (Annexe 7.3 à 7.5).

5.1 La conception naïve du langage de programmation (entretien avec Stéphanie B. et Stéphanie G.)

5.1.1 La première phase : le problème ONJOURB 1 (Compte-rendu en Annexe 7.3.1)

La réponse écrite de Stéphanie G. (**S.G.**) à l'épreuve, est si l'on considère `L` comme égal à la longueur de `X$`, proche d'une forme de la solution: Une première difficulté est que **S.G.** n'a pas donné de valeur à la variable `L` qu'elle emploie dans la ligne

```
20 LET A$ = MID$(X$,2,L+MID$(1,1))
```

Suite à cette remarque de notre part, elle remplace `L` par l'appel de fonction `LENGTH(X$)`, ce qui rend l'expression encore plus compliquée, alors qu'il aurait été possible d'écrire une ligne

```
15 LET L = LENGTH(X$)
```

qui aurait constitué ce que nous avons appelé une affectation de résultat intermédiaire. Au moment de l'observation, nous n'avons pas relevé ce choix de l'élève comme une réticence de l'élève à employer l'affectation. C'est seulement à la suite de l'analyse de l'entretien que nous avons compris que cette élève n'a pas intégré le rôle de l'affectation dans l'évolution de l'état du dispositif. Au contraire, nous avons pensé au moment de l'observation que cette ligne 20 est assez proche d'une forme correcte, qui serait :

```
20 LET A$ = MID$(X$,2,L)+MID$(X$,1,1).
```

En toute rigueur, le troisième argument du premier appel de la fonction `MID$` serait `L-1`, mais dans le BASIC utilisé, `X$`, étant de longueur `L`, `MID$(X$,2,L)` donnera le même résultat que `MID$(X$,2,L-1)`. Nous avons interprété, au moment de l'observation, l'écriture erronée comme venant d'un défaut de syntaxe, et tenté de faire évoluer la réponse vers la forme proche considérée comme correcte (ligne 20 ci-dessus). Les défauts de syntaxe auraient été :

- le parenthésage ; il manquerait une parenthèse entre `L` et `+`, et la parenthèse de la fin serait superflue ;
- l'emploi d'implicites, qui expliquerait l'absence de `X$` comme argument du second `MID$`.

La résistance des élèves aux modifications proposées, le fait que l'expression proposée par **S.G.** est effectivement correctement parenthésée de son point de vue montrent que les difficultés sont ailleurs. Elles tiennent à la compréhension qu'ont les élèves (particulièrement **S.G.**) des fonctions en jeu dans l'exercice : les écritures `MID$`, `LEN`, et `+` sont comprises comme la simple transcription d'actions : "*on prend à partir de*", "*on prend la longueur totale*", "*on ajoute*"...

Il est assez remarquable que, en ce qui concerne les nombres (entiers, puis rationnels, etc...) le passage de la compréhension des signes opératoires comme

codages d'actions à la compréhension comme fonction (par exemple, l'écriture 2 + 3 peut coder l'action d'"ajouter" deux objets et trois objets, avant d'être considérée comme une autre écriture du nombre 5) se fait au cours de la première année d'enseignement obligatoire, soit une dizaine d'années avant le niveau où nous situons nos observations. L'interprétation des écritures fonctionnelles (ou opératoires) comme codage d'actions est couramment utilisée, que ce soit en mathématiques, ou, en programmation au cours de l'analyse comme raccourci pour décrire l'effet des fonctions : dans le cas de l'exercice 2, cette analyse pourrait s'écrire: "pour obtenir le même mot, mais avec la première lettre passée à la fin, on prend la sous-chaîne du mot commençant à la seconde lettre, de longueur `L-1`, et on lui concaténe à droite la première lettre du mot". Mais il s'agit d'une métaphore qui se justifie par la lourdeur des expressions fonctionnelles dans le langage courant. `MID$(X$,1,1)`, par exemple, ne code en aucune manière une action sur la chaîne `X$`, telle que "prendre la première lettre" ; l'écriture représente seulement un objet informatique, ici une lettre, sans aucune modification de l'état de l'ordinateur. Cette écriture, si elle reste isolée, produira une erreur.

Dans la réponse de l'élève, l'écriture fonctionnelle est incluse de façon correcte dans une affectation, ce qui a contribué à ce que, au moment de l'observation, nous n'apercevions pas les difficultés de l'élève concernant l'interprétation des écritures fonctionnelles. L'observation suivante (seconde phase) nous a montré que les élèves observés n'attribuaient pas de signification à cette affectation : suite sans doute à des expériences antérieures sur machine elles savent qu'une expression `MID$...` ne peut se trouver comme instruction isolée ; c'est pourquoi **S.G.** fait précéder cette expression d'un "`LET A$=`", qui n'a pas de signification pour elle (dans le problème **RONJOUB**, elle emploie deux fois cette expression, alors qu'il s'agit de deux parties distinctes de la chaîne). Résultat d'une erreur corrigée, cette compensation n'implique pas une meilleure intégration du langage.

5.1.2 La seconde phase : le problème RONJOUB (Compte-rendu en Annexe 7.3.2)

Le programme écrit par **S.G.** à l'épreuve sur papier se compose d'une ligne identique à la ligne 20 de sa réponse au problème **ONJOURB** (Cf ci-dessus) ; nous avons indiqué ci-dessus que l'écriture fonctionnelle dans cette ligne constitue pour l'élève un codage de l'action "passer la première lettre au début", et que l'affectation qui la précède n'a pas de signification. Cette ligne est suivie de la ligne :

```
30 LET A$ = MID$(X$,1,L-1+MID$(5,1))
```

Comme nous l'avons relevé ci-dessus, l'affectation se fait à la même variable `A$` ; il n'y a d'ailleurs pas ici d'affichage de cette variable. On retrouve les mêmes défauts de syntaxe que pour **ONJOURB**, mais, après l'analyse faite ci-dessus, on comprend que cette ligne code l'action "passer la dernière lettre au début". Si la syntaxe peut être corrigée comme ci-dessus, l'obtention d'un résultat correct n'est pas immédiate, car dans les chaînes affectées à la variable `A$`, le "corps" du mot donné se trouve deux fois.

L'observation montre que, partant du programme écrit par **S.G.** à l'épreuve écrite, et des corrections apportées au programme **ONJOURB** à la phase précédente, les élèves obtiennent un programme exécutable. Bizarrement, la seconde chaîne formée à la ligne 30 est constituée de la concaténation de deux chaînes égales au mot donné amputé de la dernière lettre. A ce moment de l'observation, nous interprétons l'obtention d'une écriture syntaxiquement correcte comme un progrès. En effet, nous attendons que le "feed back" apporté par le résultat affiché par l'ordinateur permettra aux élèves d'apercevoir les erreurs de conception. Mais les élèves se montrent incapables d'interpréter le résultat affiché par l'ordinateur. Nous avons donc tenté d'orienter les élèves vers une décomposition du mot en trois parties. Mais cette tâche se révèle trop difficile pour ces élèves et il n'y a pas de réelle recherche de leur part dans la suite de l'entretien. Il aurait fallu en fait, reprendre la démarche de décomposition sur un exercice plus simple, par exemple **ONJOURB**.

5.1.3 Conclusion sur l'entretien avec S.G. et S.B.

L'entretien montre que les erreurs de **S.G.** à l'épreuve sur papier ne sont pas de simples erreurs de syntaxe facilement corrigées lors du passage sur machine : l'écriture fonctionnelle n'est pas intégrée, en ce sens qu'elle est comprise comme un codage d'actions et parallèlement, l'affectation n'a pas de signification, puisque, pour les élèves, ce sont les écritures fonctionnelles qui changent l'état du dispositif. Les difficultés des élèves sont bien celles que nous avons appelé *D.Langage* .En tant qu'observateur, nous avons, à cette époque peu de moyens pour comprendre ces difficultés. En particulier, nous pensons les élèves capables de donner une signification à une écriture entièrement fonctionnelle (n'utilisant pas l'affectation de résultats intermédiaires) ; d'autre-part, nous croyons les élèves capables d'analyser leurs écritures à l'aide du résultat produit à l'exécution. C'est pourquoi nous dirigeons ces élèves vers une amélioration en vue du respect des régles de la syntaxe. L'entretien montre que l'amélioration de la syntaxe n'a pas d'effet sur la compréhension, et que le "feed back" de l'ordinateur ne suffit pas à ce que les élèves fassent le lien entre les écritures erronées.et les résultats auxquels elles conduisent.

5.2 Les difficultés avec les ordinaux; le choix du type(Entretien avec Armelle D. et Karine C)

5.2.1 La première phase : le problème RONJOUB (Compte-rendu en Annexe 7.4.1)

Les élèves isolent ici les 3 parties du mot et les recomposent correctement pour obtenir le résultat. Elles ne rencontrent pas les difficultés (*D.Langage*) des élèves de l'entretien précédent. Le programme comporte néanmoins deux erreurs. La première à la ligne 20 (calcul du dernier caractère) est la confusion sur la position du dernier caractère que nous avons relevé plus haut et classé dans les difficultés *D.Ord.Card.* .La seconde erreur à la ligne 30 concerne le calcul de la sous-chaîne "corps", pour laquelle les élèves restent fixés sur l'expression : `MID$(X$,2,LEN(X$)-1)`. Le troisième argument est erroné ; il faudrait `LEN(X$)-2`.

Au début de l'entretien, les élèves analysent le résultat de l'exécution et corrigent la première erreur, puis, un succès partiel étant obtenu, appliquent la même correction à la ligne 30 (remplacement de `LEN(X$)-1` par `LEN(X$)`). Comme observateur, cette seconde correction nous a paru ne pas avoir de cohérence ; pourtant, pour les élèves, dans la ligne 20, comme dans la ligne 30, `LEN(X$)_ 1` **code la séparation du dernier caractère**, et il est donc logique d'appliquer la même correction aux deux lignes. Au moment de l'observation, nous analysons au contraire cette écriture comme cohérente pour les élèves avec un effet de bord qu'elles supposeraient causé par l'extraction de la dernière lettre à `X$` à la ligne 20 : cette ligne aurait été comprise par les élèves comme retirant effectivement cette dernière lettre, en plus d'affecter à `A$` cette dernière lettre Il semble clair, à la lecture du compte-rendu, que c'est la ligne 30 elle-même, et particulièrement le troisième argument de `MID$` qui est en cause : ce troisième argument est en effet compris comme la traduction directe de "**on retire le dernier caractère**" (*D.Sem.Cha.*).

Nous retrouverons dans la suite de cette thèse d'autres observations où, ayant donné la position de départ comme second argument, l'élève donne comme troisième argument une écriture qu'il considère comme **une indication supplémentaire pour le calcul de la sous chaîne**. Cette indication a un statut informatique flou ; il se trouve qu'ici, alors que pour les élèves cette écriture est d'abord la traduction d'une action, elle est bien syntaxiquement un nombre, d'où notre difficulté, comme observateur, à comprendre la signification que les élèves lui attachent. Si l'on cherche à faire correspondre cette écriture avec notre grille de difficultés, on s'aperçoit que trois difficultés sont concernées. Il s'agit d'une mauvaise compréhension du troisième argument de la fonction sous-chaîne (`MID$`), donc *D.Sem.Cha.*. est concernée. Une écriture fonctionnelle est comprise comme la traduction directe d'une action, ce que nous faisons correspondre avec *D.Langage.* La fonction longueur intervient dans l'écriture, et l'erreur (vite corrigée) sur la recherche du dernier caractère en ligne 20 a déjà montré que la signification de `LEN(X$)` comme argument dans `MID$` est loin d'être claire, les confusions étant liées à la double signification de la longueur : comme

ordinal, c'est la position du dernier caractère ; comme cardinal, c'est le nombre de caractères de la chaîne. La soustraction, dans `LEN(X$)_ 1` n'est comprise ni comme une opération sur des cardinaux, ni comme une translation sur des ordinaux, mais dans un sens plus primitif d'opération sur des objets (ici des caractères) faisant en quelque sorte la synthèse des aspects cardinal et ordinal. *D.Ord.Card.* est donc également concernée.

5.2.2 La seconde phase : résolution d'un problème prolongeant le problème NOTES (Compte-rendu en Annexe 7.4.3

Dans l'exercice **NOTES** à l'épreuve sur papier, tous les élèves avaient été surpris par l'homothétie ordinale, et nous avions noté que ceux qui trouvaient une expression de cette homothétie le faisaient suite à une recherche empirique, sans mettre en oeuvre des connaissances acquises en mathématiques sur les fonctions affines. **A.D.** et **K.C.** font partie de ces élèves. Nous avons voulu leur proposer, devant l'ordinateur, un exercice du même type (le calcul du nième caractère de l'alphabet). Les élèves adaptent spontanément le rapport d'homothétie, et, par essai mental, le terme constant. A la suite de cet entretien, nous pensons que ces élèves ont bien intégré la construction de l'homothétie ordinale. Les difficultés (*D.Ord.Card.*) rencontrées au cours de l'épreuve sur papier ne se manifestent pas ici.

Il n'y a pas ici, contrairement à l'exercice 4, d'enchaînement complexe de situations ; les élèves n'ont pas de difficulté à analyser le fonctionnement du programme ; le processus d'essai-erreur ne nécessite pas d'exécution en machine, et se révèle producteur.

5.2.3 La troisième phase : résolution du problème CHIFFRE et prolongement (Compte-rendu en Annexe 7.4.3)

Nous avions voulu observer ici d'une part la difficulté concernant la position du dernier caractère (*D.Ord.Card.*) et la difficulté concernant le choix du type (*D.Type*). Nous constatons que la position du dernier caractère fait toujours problème, même après qu'elle ait été corrigée au cours de la résolution du problème **RONJOUB**. Elle paraît donc liée à des incompréhensions assez profondes de la notion d'ordinal, mais aussi du rôle des fonctions.

Concernant les questions de type, il se confirme que le choix d'un type numérique est bien délibéré, et ne résulte pas d'une négligence. Nous indiquons aux élèves l'emploi de variables chaînes pour les nombres. Cet emploi est rendu valide par l'essai en machine, mais sans réelle compréhension de ses raisons. Nous revenons sur cette question en proposant un problème où il faut comparer une variable chaîne à une constante chiffre. Cette comparaison implique l'écriture du chiffre constant entre guillemets (les guillemets délimitent les constantes chaîne en BASIC). Les élèves ont rencontré les guillemets comme délimiteurs de constantes chaînes comportant des lettres ou des étoiles ; ils ont compris que les guillemets servent à différencier la constante chaîne et l'identificateur de variable composé des mêmes caractères. Mais ici la situation est différente, puisque un nombre est bien compris par l'interpréteur comme une constante (numérique), et les élèves savent qu'un identificateur de variable ne peut s'écrire comme un nombre. Les guillemets sont nécessaires ici, car un nombre et la chaîne de caractères formée des chiffres de ce nombre dans l'écriture décimale sont des entités fondamentalement différentes d'un point de vue informatique, bien qu'ils soient tous les deux des constantes, et aient quasiment la même forme externe, C'est cette différence que les élèves, conditionnés à assimiler un nombre à son écriture décimale, ont des difficultés à percevoir.

5.2.4 Conclusion sur les entretiens avec A.D et K.C.

Contrairement aux élèves de l'observation précédente, **A.D.** et **K.C.** maîtrisent suffisamment les éléments généraux du langage. La décomposition de la tâche, et l'affectation des résultats intermédiaires leur permettent un début d'analyse du résultat obtenu à l'exécution. Mais, quand nous leur demandons de faire fonctionner le programme "à la main", c'est à dire de calculer pas à pas le résultat des fonctions en jeu, ou d'isoler une partie du programme, les élèves marquent une forte résistance à

l'analyse élément par élément de leur programme

Si ces élèves ont suffisamment intégré le langage pour utiliser à bon escient l'affectation et les écritures fonctionnelles, les difficultés concernant le langage réapparaissent pour contribuer à la fixation des élèves sur des erreurs dans les calculs d'ordinaux et de cardinaux (problème **ONJOURB**), que nous avons également reliées à une difficulté concernant l'interprétation de la structure "chaîne de caractères". Nous verrons d'autres exemples où une même écriture peut être reliée à plusieurs types de difficultés ; il se peut dans ce cas, qu'une difficulté domine chez l'élève, mais il n'est guère possible de le savoir par une observation ponctuelle ; dans d'autres cas, plusieurs conceptions erronées peuvent coexister et contribuer à la fixation sur une écriture erronée. On constate à la lecture de l'entretien, que l'interrogation directe des élèves donne peu d'indications. De façon générale, au cours de la première phase, nous ne parvenons pas, comme observateur, à situer l'origine réelle des difficultés des élèves.

Nous avions, pour l'épreuve sur papier, classé *D.Ord.Card* les difficultés de résolution dans le problème **NOTES**. Les élèves observées avaient rencontré ces difficultés à l'épreuve sur papier ; elles montrent ici qu'elles ont bien intégré la résolution de ce type d'exercice.

La troisième phase montre que la question du type de la variable se pose réellement pour les élèves. Elle n'est pas entièrement résolue, et l'apparition de nombres sous forme de constantes chaînes révèle un nouvel aspect de cette difficulté.

5.3 L'installation d'un S.R.T. relatif au traitement des chaînes dans un S.R.T. opératoire plus général (Entretien avec Arnaud P.) Compte-rendu en Annexe 7.4.3

Comme les deux élèves précédents, **A.P.** domine les éléments généraux du langage suffisamment pour apercevoir la nécessité d'isoler trois sous-chaînes et de les restituer ; par contre il n'utilise pas l'affectation pour les résultats intermédiaires, et se contente d'un affichage sans passage à la ligne pour les restituer. Le premier programme qu'il écrit comporte trois erreurs :

- une soustraction entre une chaîne (`A$`) et un nombre comme troisième argument de la sous-chaîne "dernier caractère".
- un troisième argument erroné dans le calcul de la chaîne corps (`LEN(A$)-1` au lieu de `LEN(A$)-2` , c'est-à-dire la même erreur que **K.C.** et **A.D.**)
- la restitution des éléments du résultat dans l'ordre initial (sans l'interversion demandée dans l'énoncé).

La première erreur est une erreur de syntaxe ; l'élève la corrige suite à notre intervention, en `LEN(A$)_1`. Nous n'apercevons pas lors de cette observation que l'élève, comme ceux que nous avons observé précédemment, code en fait l'action "séparer le dernier caractère"; notre intervention conduit donc à une syntaxe correcte, mais il n'est pas certain que la compréhension progresse.

A la suite de cette correction, le programme est correct syntaxiquement, mais comporte les mêmes erreurs que le programme initial de **K.C.** et **A.D.**, plus la troisième erreur (restitution sans interversion). A la différence de **K.C.** et **A.D.**, **A.P.** tire un grand profit du résultat obtenu à l'exécution : il isole les trois parties en les faisant afficher chacune sur une ligne, corrige la restitution sans interversion, puis l'erreur sur la position du dernier caractère. Il ajuste de façon pratiquement autonome le troisième argument dans le calcul de la chaîne corps, en raisonnant sur un exemple.

Cet élève produit donc des erreurs tout à fait semblables à celles que nous avons rencontré chez les autres élèves, avec les mêmes difficultés sous-jacentes (conjonction de *D.Langage, D.Sem.Cha.*, et *D.Ord.card.)*. La différence est que la correction des erreurs est beaucoup moins laborieuse. La pratique de mise au point interactif (déboguage) lui permet de rechercher l'écriture ayant causé l'erreur, et de la modifier en conséquence. La brièveté de l'entretien ne permet pas d'en savoir beaucoup plus. Il est raisonnable de penser que **A.P.** a une meilleure maîtrise du "déboguage" que **K.C.** et **A.D.** parce qu'il intègre mieux les éléments généraux du langage. Nous nous trouverions dans ce cas en présence d'un sujet disposant d'un **S.R.T.** général adéquat

et devant y intégrer un **S.R.T.** correspondant à un type de données nouveau : des ajustements sont nécessaires, mais ils n'impliquent pas une remise en cause trop profonde du **S.R.T.** existant.

6. Résumé du chapitre, et conclusions

Le chapitre porte sur l'étude de la résolution de problèmes impliquant l'emploi de chaînes de caractères, avec une simple structure séquentielle. Nous avons prévu quatre classes de difficultés, et proposé aux élèves, d'abord sur papier, puis devant l'ordinateur, une épreuve (EPR1) comportant ces difficultés :

- *D.Langage*: les difficultés liées à une conception naïve du langage de programmation; elles concernent les éléments généraux du langage (affectation, écritures fonctionnelles) mais pourraient avoir été masquées lors de la résolution de problèmes portant seulement sur des données numériques et/ou la gestion de l'écran.
- *D.Sem.Cha.* : Les difficultés propres à la structure de chaîne.
- *D.Ord.Card.* : les difficultés liées à la conception des ordinaux, et à la coordination ordinaux-cardinaux.
- *D.Type* : les difficultés liées au typage des objets.

Les difficultés des élèves

Nous reprenons, difficulté par difficulté, les conclusions apparues dans le dépouillement des réponses des élèveset celles apparues avec l'analyse des entretiens. Puis nous tirons des conclusions concernant notre intervention pédagogique auprès des élèves au cours des entretiens.

- **Les difficultés *D.Langage*** : elles sont présentes chez la moitié des élèves pour le problème **ONJOURB** à l'épreuve écrite, et les conduisent à des écritures syntaxiquement erronées ; à la différence de ceux qui réussissent, ces élèves n'emploient pas l'affectation des résultats intermédiaires comme décomposition du problème, et par conséquent, leurs écritures sont complexes ; nous avons fait l'hypothèse que, chez ceux qui réussissent, l'acquisition de l'affectation a progressé de pair avec une meilleure compréhension des écritures grâce à la modularité permise par l'affectation des résultats intermédiaires. Il se confirme lors de l'entretien avec deux des élèves ayant rencontré ces difficultés dans **ONJOURB** (paragraphe 5.1) que les écritures erronées ne sont pas de simples erreurs de syntaxe; elles ont une signification pour les élèves : les écritures fonctionnelles codent des actions sous une forme proche de métaphores en langage naturel, et l'affectation n'est pas comprise comme modifiant l'état du dispositif ; quand l'affectation est présente, elle correspond à une exigence syntaxique et n'a pas de signification pour l'élève.
- **Les difficultés *D.Sem.Cha.***: elles paraissent ici difficiles à séparer des difficultés *D.Langage* pour les élèves qui rencontrent ces dernières. Pour d'autres élèves, la signification des arguments de la fonction `sous-chaîne` peut être erronée, mais ceci est lié également à des difficultés *D.Ord.Card.* (voir ci-dessous). Les élèves n'emploient pas la concaténation, car ils utilisent l'affichage sans passage à la ligne pour former le résultat directement à l'écran. Ceci correspond à la pratique de la classe dans les apprentissages antérieurs et, a posteriori, nous regrettons que l'énoncé n'ait pas davantage insisté sur la nécessité de calculer le résultat avant de l'afficher, ce qui aurait conduit à des observations sur la concaténation.
- **Les difficultés *D.Ord.Card.***: les réponses des élèves au problème **NOTES**, montrent qu'ils ne réinvestissent pas les connaissances qu'ils auraient pu acquérir en mathématiques dans le domaine des fonctions affines. Nous attribuons ceci au fait que la relation affine concerne dans le problème **NOTES** des ordinaux, alors que les connaissances des élèves en mathématiques sont principalement dans le domaine des cardinaux. Cependant un entretien avec deux élèves ayant un niveau dans la moyenne de la classe (paragraphe 5.2.2) permet de constater que, après résolution du problème **NOTES**, la fonction affine sur les cardinaux est bien intégrée, car réinvestie sans difficulté dans un exercice analogue. Nous pouvons donc faire

l'hypothèse qu'un travail spécifique sur les fonctions ordinales permet aux élèves de progresser.
L étant la longueur d'une chaîne donnée, on trouve chez de nombreux élèves, y compris parmi ceux qui n'ont pas rencontré les difficultés *D.Langage* la position L-1 (au lieu de L) pour le dernier caractère. Cette erreur nous a paru liée à une mauvaise coordination entre ordinaux et cardinaux : en effet, comme cardinal, L c'est "toute la chaîne", donc le dernier caractère "commencerait juste avant", d'où la position L - 1. On rencontre également la longueur L-1 (au lieu de L-2) pour la sous-chaîne sans le premier et le dernier caractère qui s'explique plus difficilement. En fait, l'écriture L-1 y compris pour des élèves dans une bonne moyenne, a le sens du codage d'une action, à relier avec la façon dont les élèves concernés par *D.Langage* considèrent les écritures fonctionnelles : en effet dans les écritures concernées, il s'agit bien de séparer le dernier caractère : L c'est *"tout le mot"* et -1, c'est *"séparer le dernier"*. Une même erreur d'écriture se trouve donc concerner trois difficultés : *D.Ord.Card.* puisqu'il y a une mauvaise coordination entre ordinaux et cardinaux , *D.Sem.Cha.* puisque l'erreur apparaît dans l'écriture d'arguments de la fonction sous-chaîne dont le rôle n'est pas encore bien précis pour les élèves, et *D.Langage* comme nous l'avons montré ci-dessus. L'écriture erronée provient-elle d'une seule parmi les trois, l'analyse ne permettant pas d'en décider, ou y a t'il une conjonction des trois difficultés qui conduirait à l'erreur. Nous penchons pour la seconde hypothèse : en effet, les élèves que nous avons observé en entretien (sous paragraphes 5.2 et 5.3) ne rencontrent pas la difficulté *D.Langage* en dehors de l'écriture de ces arguments de la fonction sous-chaîne. Nous retiendrons que des élèves peuvent avoir été intégré le langage suffisamment pour écrire des programmes généralement corrects, mais que des conceptions naïves du langage peuvent rester présentes chez eux et réapparaître à l'occasion de difficultés nouvelles (ici les difficultés qui apparaissent avec la structure de chaîne et les ordinaux).
- **Les difficultés *D.Type*** : le problème **CHIFFRE** met bien en évidence la difficulté à choisir le type d'une variable en fonction d'une abstraction implicite des données, et non en fonction de la nature intuitive des données (dépouillement 4.3 et entretien 5.2.2). L'entretien met en évidence d'autres difficultés liées, concernant particulièrement l'écriture des constantes.

Lors des apprentissages antérieurs, la notion de chaîne de caractères a été présentée aux élèves concernés par EPR1. Par contre, les difficultés que nous avons recensées ci-dessus ont été peu sollicitées: en particulier, les sous-chaînes calculées dans les exercices antérieurs à l'épreuve ont toutes comme origine le premier caractère: ainsi le calcul des ordinaux n'est pas en jeu dans ces exercices, et une réponse correcte peut être obtenue malgré des conceptions erronées.

Nos interventions pédagogiques au cours des entretiens.

Les entretiens permettent d'observer nos interventions pédagogiques auprès des élèves, et d'en estimer la pertinence. Il nous semble qu'à cette époque, nous avions très peu de moyens pour interpréter les erreurs des élèves et analyser leurs difficultés. Nous avons montré ci-dessus qu'une même écriture erronée peut résulter de la conjonction de difficultés de nature différente, et qu'une écriture apparemment correcte peut ne pas avoir de signification pour l'élève, mais être le résultat d'une correction purement syntaxique. Ceci explique que, sans réflexion préalable approfondie, l'analyse des difficultés des élèves n'est pas possible. De façon générale, nos interventions, telles qu'ont peut les observer dans le dépouillement, ont visé à obtenir un programme correct syntaxiquement, par rappels des régles de syntaxe, puis à faire réfléchir les élèves sur les résultats obtenus à l'exécution, de façon à tenter de leur faire apercevoir les défauts de conceptions.

Ces interventions n'ont pratiquement pas d'efficacité avec le premier groupe d'élèves confrontées à des difficultés *D.Langage* très importantes, et sont relativement pertinentes auprès du dernier élève qui nous a semblé disposer d'un **S.R.T.** général suffisant, le **S.R.T.** particulier aux chaînes de caractères étant en voie d'installation;

le second groupe d'élèves occupe de ce point de vue une position médiane: nos interventions aboutissent, mais nous avons noté les difficultés des élèves à analyser le résultat obtenu à l'exécution.

Pour des élèves confrontés à des difficultés *D.Langage* très importantes, les corrections syntaxiques sont trop locales, et ne remettent pas en cause leur conception du langage; de même, l'interprétation des résultats obtenus à l'exécution supposerait un minimum de compréhension du langage. Il est clair, à la suite de ce chapitre, que en dehors du troisième élève, une intervention pédagogique ne peut faire l'économie de la connaissance des difficultés spécifiques des élèves auxquels elle s'adresse. Si les entretiens montrent les limites de l'intervention pédagogique que nous pratiqué au cours des entretiens, les conditions d'une intervention plus efficace restent à définir.

Chapitre 7 Intervention des plans relatifs aux traitements

Résolution par une classe de Seconde de problèmes de "recherche-translation" sur une chaîne de caractères Interactions, dans le S.R.T., des plans relatifs aux objets et des plans relatifs aux traitements

Nous avons observé au chapitre précédent les conduites d'élèves confrontés à la résolution de problèmes "directs", de façon à comprendre et à classifier les conceptions des élèves portant sur des objets représentables à l'aide du type "chaîne de caractères", à travers les difficultés que leur créent ces conceptions dans la résolution de problèmes de programmation. Il s'agit dans ce chapitre, d'examiner l'intervention de plans concernant les traitements, et leur interaction avec les plans relatifs aux données étudiés au chapitre précédent. C'est pourquoi nous introduisons dans les problèmes, la nécessité d'une action complexe (c'est-à-dire une action imposant une rupture de séquentialité), dont l'acquisition est en cours chez les élèves.

Nous avons, comme au chapitre précédent, fait passer aux élèves une épreuve sur papier d'une heure (EPR2), et procédé à un entretien. Le sous-chapitre 1 précise nos hypothèses, le type d'action complexe, et la classe de problèmes à partir de laquelle nous menons les observations. Le sous-chapitre 2 décrit et analyse l'enseignement reçu par les élèves, et présente l'épreuve. Aux sous-chapitre 3 et 4, nous analysons respectivement le réponses aux questions 1 et 2 de l'épreuve. Puis nous analysons le dépouillement de l'entretien (sous-chapitre 5). Le sous-chapitre 6 contient les conclusions du chapitre. On trouvera en annexe à ce chapitre, un rappel de l'énoncé de l'épreuve, un dépouillement des réponses des élèves, et de l'entretien.

1. Hypothèses quant aux difficultés des élèves, et choix d'observation

1.1. Les difficultés propres aux actions complexes : intervention des conditions, et des ruptures de séquentialité

Nous avons présenté au chapitre 4 (§ 1.4) les principales structures algorithmiques correspondant aux actions complexes (alternative et itération), et rapporté des difficultés observées dans des enseignements de ces structures à des débutants.

- En premier lieu, ces deux structures impliquent la présence d'une **condition** et nous avons vu que la construction et/ou l'interprétation d'une telle condition peuvent ne pas être immédiates. Les *précurseurs logiques* (en particulier la maîtrise de la négation) sont souvent absents, et les élèves n'attribuent pas un sens suffisamment précis aux conditionnelles portant sur des *objets non encore envisagés comme calculables* . Dans ce cas, les élèves prêtent au langage de programmation des propriétés du mode de communication habituel, ce qui est à mettre en relation avec la *conception naïve du langage* que nous avons étudiée au chapitre 5 (§ 1.1) pour des programmes sans structure algorithmique complexe. [81]
- De plus, l'alternative et l'itération impliquent des **ruptures de**

[81] Donnons un exemple de ce comportement, observé en fin de classe de seconde. En fin d'année, les élèves ont à à écrire un programme permettant à l'utilisateur de jouer au jeu du pendu. Un élève se donne pour tâche l'écriture d'une condition évaluant si un caractère entré par l'utilisateur appartient bien au mot à trouver ; il affecte ce caractère à la variable `LETTRE` de type caractère puis écrit une alternative commençant par `IF LETTRE = TRUE` (`TRUE` est en **PASCAL** la constante booléenne ayant la valeur vraie) et obtient par conséquent à la compilation une erreur de type qu'il ne comprend pas. Il explique qu'il a voulu traduire dans le langage de l'ordinateur la phrase "*si la lettre est bonne*", et que l'ordinateur aurait dû comprendre.

séquentialité. Le mode d'interprétation de ces ruptures de séquentialité dans chacune de ces structures, et par conséquent leurs difficultés spécifiques, paraissent, dans l'état actuel des recherches, différer sensiblement. Dans le cas d'alternatives en chaîne, l'interprétation passe par la *collecte d'informations* permettant de connaître les conditions requises pour une issue donnée du programme ; [GREEN77], article présenté au chapitre **2** §2, montre que cette collecte peut ne pas être immédiate, et dépend fortement des possibilités de codage des alternatives en chaîne dans le langage utilisé. La signification d'une itération est donnée quant à elle par une entité logique : *l'assertion invariante* qui exprime une relation entre les variables itératives (c'est à dire celles dont la valeur varie au cours de l'itération). indépendante du point atteint dans l'itération. Nous avons indiqué (chapitre **5** § 1.4.5) que la recherche d'une assertion invariante comme préalable à la construction de l'itération, telle qu'elle est proposée par [ARSAC 80] nous semble être du niveau du raisonnement par récurrence, et donc dépasser le niveau d'enseignement que nous visons. Il n'en reste pas moins que, partant d'une conception de l'itération comme action répétée, les élèves doivent, pour donner un sens à cette itération, et donc contrôler leur anticipation, dégager des invariants logiques et les coordonner avec leurs conceptions du traitement par le dispositif. Ainsi, les élèves observés par R. SAMURCAY ([SAMURCAY 85] article présenté au chapitre **2** §4) donnent un sens à la variable `compteur` : ils peuvent à la fois dégager sa signification indépendamment du point atteint dans l'itération, et imaginer un fonctionnement du dispositif rendant compte de cette signification. Par contre ces élèves rencontrent des difficultés avec d'autres variables cumulatives, comme par exemple le résultat partiel dans un calcul itératif : ils peuvent avoir des difficultés à établir un lien entre l'action répétée sur ces variables et l'état des variables, dans ce cas, par exemple, la valeur initiale est reprise à chaque étape du calcul ; également, à partir d'une situation vue en mathématique (l'exponentiation), ils peuvent ne pas parvenir à dégager un schéma de calcul compatible avec le dispositif (ce qui indique que le traitement mathématique et le traitement pour un dispositif automatique appartiennent à des plans que les élèves ne parviennent pas à différencier et/ou à coordonner).

1.2 Nos choix d'observation

1.2.1 Choix d'une action complexe : l'itération à un point de sortie choix d'une classe de problèmes : la recherche d'informations dans une chaîne

Le but de ce chapitre, est d'étudier comment l'acquisition d'une structure complexe (alternative,itération), peut s'insérer dans le cadre général que nous avons proposé au chapitre **3** comme explicatif des difficultés des débutants. Nous ne sommes pas bien sûr en mesure de faire une étude exhaustive, et nous avons dû faire un choix ; au moment de commencer l'observation expérimentale rapportée dans ce chapitre, nous avions connaissance seulement de recherches concernant l'itération ([SAMURCAY 85] et les recherches que nous avons présentées en annexe au chapitre **5**) et il nous a donc semblé plus facile de faire des hypothèses dans le domaine de l'itération. Nous avons donc fait le choix d'étudier l'acquisition de l'itération plutôt que de l'alternative. Les recherches nous ont montré également que les nombreuses difficultés de l'itération interférent et conduisent souvent à l'échec. Nous avons rappelé ces difficultés au chapitre **5** (§ 1.4.5) ; en particulier, s'il existe plusieurs variables itératives, la question de la coordination de leur évolution dans le corps de boucle, et la question de leur intervention dans la condition d'arrêt est non-triviale.

Nous avons donc décidé de nous en tenir à une classe de problèmes où les difficultés sont bien délimitées: la recherche itérative d'informations dans une chaîne. Nous présenterons plus loin dans le chapitre comment on peut construire des problèmes sur ce thème sans accumuler les difficultés; nous précisons au paragraphe suivant comment cette classe de problèmes s'insère dans notre problématique générale : étudier les conduites résultant chez le sujet d'un système de représentation et de

traitement (**S.R.T.**) ne différenciant pas les actions du dispositif de celles qui sont opérées dans le contexte familier.

1.2.2 Observation des conduites liées à la conception que se font les élèves de la recherche d'informations dans un texte

La notion familière la plus proche de la chaîne de caractères est la notion de texte. Un texte se distingue des chaînes de caractères par le fait qu'il a une (des) signification(s) dans un contexte donné. L'ordinateur, ne considérant pas les significations, traite des chaînes et non des textes. Ce traitement formel est utile dans la mesure où il s'appuie sur un agencement particulier des éléments de la chaîne matérialisant certaines significations (ainsi un "traitement de texte" est capable de reconnaître un groupe de caractères comme un "mot" parce qu'il existe des caractères séparateurs: espace, ponctuation...). Nous considérerons comme un "*texte*" toute chaîne de caractères de laquelle un traitement formel permet d'extraire une information significative. On peut faire l'hypothèse que, pour un débutant en informatique, les nécessités du traitement formel par un dispositif informatique ne sont pas perçues. La recherche d'informations dans un texte fait en effet partie des activités familières et l'opérateur humain met en jeu dans cette recherche de façon indistincte des éléments sémantiques et des éléments formels : il n'a pas conscience de se livrer à un calcul[82] ; par contre, le traitement formel par un dispositif informatique impose **un algorithme de parcours**, et **la transposition sous forme d'entités calculables** des objets mis en jeu par cet algorithme. Par exemple, dans le cas du parcours d'un texte, il faudra choisir un sens de parcours et construire l'itération correspondante ; cette itération fera intervenir un index repérant le caractère lu ; il faudra aussi, compte-tenu de la structure de chaîne, et des nécessités de l'itération, concevoir cet index comme ordinal, alors qu'un opérateur humain sait passer d'un caractère au suivant ou au précédant sans numéroter les caractères. Nous introduisons ainsi la *distance* entre le traitement habituel et le traitement par un dispositif automatique qui nous paraît en mesure de révéler l'existence de conceptions n'intégrant pas les spécificités du traitement par le dispositif (voir chapitre 3).

1.2.3 Observation de l'intervention de difficultés repérées dans un cadre non itératif

Dans le **S.R.T.** de l'élève, indépendamment des conceptions correspondant à des actions complexes, les conceptions propres au traitement des chaînes peuvent se révéler inadéquates; nous avons tenté au chapitre 6 une classification des difficultés en résultant, observées à travers la résolution de problèmes n'impliquant pas un traitement complexe, et souligné qu'une réponse erronée de l'élève s'analyse le plus souvent comme conjonction de plusieurs fausses conceptions: conception naïve du langage de programmation (*D.Langage*), conceptions erronées dues à la non-intégration des fonctions sur les chaînes (*D.Sem.Cha.*), conceptions relatives aux ordinaux et aux cardinaux (*D.Ord.Card.*)[83].

Notre hypothèse est que, dans la classe de problèmes que nous proposons, il y a **intervention conjointe** des plans, c'est-à-dire qu'un comportement observé pourra s'analyser comme résultant conjointement d'un plan du **S.R.T.** correspondant au traitement complexe demandé, et d'un plan étudié au chapitre 5 (plus spécifique des données): par exemple, nous avons vu ci-dessus que l'intervention d'une *condition* (interprétée par l'élève dans un plan propre à un traitement complexe) sur des objets *non encore envisagés comme calculables* fait ressortir *D.Langage* (qui est relative aux

---

[82] La recherche d'un mot dans un texte court par l'opérateur humain fait appel à des mécanismes de reconnaissance de forme par appréhension globale plutôt qu'à un processus algorithmique. Dans un texte plus long, un algorithme de déplacement dans le texte pourra exister, mais le texte sera en fait découpé en entités courtes où l'appréhension globale fonctionne. Une autre spécificité du traitement par l'opérateur humain est l'utilisation d'une structuration du texte s'appuyant sur des éléments sémantiques et/ou syntaxiques pour abréger le processus algorithmique.

[83] Par contre, dans un souci de simplification, nous laissons de côté l'absence de prise en compte du type de l'objet (*D.Type*) qui paraît relativement indépendante des autres conceptions.

données); également, l'*index* dont nous parlons au paragraphe ci-dessus est un ordinal: la façon dont les élèves le considèrent sera influencée à la fois par la façon dont il intervient dans leur conception du traitement itératif, et par leur conception des ordinaux (*D.Ord.Card.)*[84].. Nous avons rappelé au chapitre **3** que, suivant [HOC 87], dans la résolution de problèmes s'appuyant sur la modélisation d'une situations familière, le sujet fait intervenir des *plans* directement déduits du **S.R.T.** correspondant à la situation familière; construisant, au cours de la résolution un **S.R.T.** correspondant au dispositif, il adapte à ce **S.R.T.** des éléments du **S.R.T.** correspondant à la situation familière, mais, malgré l'adaptation, ceux-ci peuvent rester marqués par des caractéristiques de la situation familière, particulièrement dans le cas où ces caractéristiques ne sont pas contradictoire avec la solution qu'il produit.

1.2.4 Observation du rôle de l'analyse en langage naturel

Parallèlement à l'introduction d'actions complexes comme l'itération, l'enseignement auprès de débutants comporte des aspects **méthodologiques**: l'aspect méthodologique essentiel en classe de seconde est l'analyse du problème comme préalable à la programmation. Cette analyse se présente le plus souvent comme l'expression en langage naturel d'un processus de résolution que l'élève doit ensuite coder dans le langage de programmation. Le programme officiel précise : "*On attachera donc la plus grande importance à l'analyse du problème comme préalable à la programmation. On évitera qu'elle ne soit considérée par les élèves que comme une paraphrase en français courant du programme à écrire : elle doit évidemment précéder la programmation et non pas la suivre...*" [85] La classe que nous avons observée ne fait pas exception comme on le verra dans la présentation du cours et des exercices que l'on trouvera en annexe 1 à ce chapitre.

S'agissant de débutants, cette phase d'analyse pose de nombreuses questions : tout d'abord, il est clair que cette analyse est une transposition du processus descendant de la programmation structurée ; en effet l'insistance sur son antériorité à la programmation indique qu'elle ne constitue pas une simple "explication" par laquelle l'élève montrerait qu'il a compris la généralité de son programme. L'analyse constitue bien l'étape essentielle de la construction du programme. Or l'analyse telle qu'envisagée dans le cadre d'un processus descendant répond à la nécessité d'un "plan d'ensemble", préparant la décomposition en modules, et suppose donc une tâche suffisamment complexe pour justifier cette décomposition. Nous avons vu dans [HOC 77] rapporté au chapitre **2** (§1), que, dans le cas d'un problème complexe impliquant une décomposition en modules et la coordination des modules, le programmeur ne peut produire une analyse efficace que s'il est capable d'imaginer une décomposition en modules directement compatible avec les contraintes du dispositif ; dans le cas contraire, toute modification sur un module a des incidences sur les autres, et remet en cause l'analyse. L'analyse préalable à la programmation est donc pertinente à partir d'une certaine complexité du problème, et de certaines capacités du programmeur. Qu'en est-il pour des problèmes où le nombre de modules est faible (un ou deux, pour les problèmes posés en initiation), et pour des apprentis programmeurs qui découvrent les contraintes du dispositif ?

Nous poserons la question de la façon suivante : l'analyse comme rédaction en langage naturel préalable à la programmation, a t'elle une pertinence supplémentaire par rapport à la simple "explication" (éventuellement postérieure à la programmation),

---

[84] Nous parlons ici d'intervention conjointe des plans du **S.R.T.**car nous nous limitons à une observation ponctuelle ; il est clair qu'une question plus générale (et plus intéressante) serait celle de l'interdépendance des plans dans la genèse des acquisitions : quels déséquilibres l'introduction d'un mode de traitement nouveau crée t'il dans des plans du S.R.T. relatifs aux données ? Comment l'intervention d'un type nouveau nécessite t'il un réajustement des conceptions relatives au traitement ? Nous aborderons ces questions respectivement aux chapitres **9** et **10**.

[85] Commentaire du programme de seconde Arrêté du 31 mai 1985 [CNDP 87]

en aidant l'élève à se construire un système de représentation et de traitement adapté à la résolution du problème et compatible avec le dispositif informatique ? Ou au contraire piège t'elle l'élève en l'enfermant dans un procédé de résolution trop éloigné d'un dispositif automatique ? Nous tenterons un premier éclairage sur cette question en proposant, dans un problème, d'exprimer une analyse avant l'écriture d'un programme (problème 2 ci-dessous).

2. Le contexte de l'épreuve

2.1 Les élèves ont reçu un enseignement portant sur l'itération et sur l'analyse

Nous avons fait passer cette épreuve (EPR2) dans la même classe que l'épreuve du chapitre précédent (EPR1). Nous avons résumé au chapitre 6 (§ 2), les cours et exercices antérieurs à EPR1. On trouvera en annexe 7 §1 (Volume Annexes) le détail des cours et exercices pendant les 4 mois séparant EPR1 et EPR2 Nous analysons brièvement les aspects principaux de cet enseignement: **l'introduction de l'itération** (qui est nouvelle pour les élèves), la façon dont le **type des objets** est pris en compte, et **l'initiation à l'analyse** à propos de laquelle nous avons posé des questions ci-dessus (1.3).

2.1.1 L'enseignement de l'itération

2.1.1.1 Les choix du professeur

L'itération est d'abord présentée comme la répétition d'une action et l'activité des élèves consiste à écrire des itérations contrôlées par un compteur (`FOR...NEXT`) : dans les premiers exercices, l'itération est la répétition d'une action (l'affichage de valeurs) sans influence sur l'état du dispositif ; le "compteur" est la seule variable qui évolue, sa gestion étant à la charge du langage, et non du programmeur. Avec le problème de la somme de `N` nombres entrés au clavier, apparaît une variable cumulative (la somme partielle) dont l'élève doit assurer la gestion, c'est-à-dire concevoir l'action répétée sur cette variable et son initialisation.

La forme `TANT QUE`, est vue seulement en cours ; elle est présentée sur un exemple (le contrôle d'une entrée au clavier) où la difficulté première est la rédaction de la condition de continuation. Le professeur indique comment cette forme peut être construite avec des instructions de branchement (`aller en`, qui s'écrit `GOTO` en BASIC). De façon concrète, le codage en machine par les élèves d'itérations de la forme `TANT QUE` imposerait l'usage du `GOTO`, car la version de BASIC utilisée par les élèves comporte seulement les avertisseurs `FOR...NEXT` ; il semble que le professeur soit réticent à cette utilisation du `GOTO`, car les programmes codés par les élèves en travaux dirigés utilisent seulement les avertisseurs `FOR...NEXT`.[86]

2.1.1.2 Les difficultés des élèves

Pour une majorité des élèves, l'introduction d'une variable cumulative avec le problème de la somme de `N` nombres entrés au clavier constitue une difficulté importante. Ensuite, le programme de conversion binaire décimal révèle une accumulation de difficultés y compris chez les élèves qui ont réussi aux exercices précédents : compréhension du problème, calcul des caractères d'un chaîne et surtout,

---

[86] Cette intervention du `GOTO` à titre *d'explication* dans le cours peut traduire une volonté de montrer le **fonctionnement** du dispositif au cours de **l'exécution** de l'itération ; on peut penser, en tant qu'observateur, que cette "monstration" aurait été utilement complétée par un codage en machine au cours des Travaux Pratiques. Comme il n'en a rien été, on suppose une réticence du professeur devant l'utilisation effective du `GOTO`. Les élèves, si on les autorise à utiliser cette instruction de branchement pour coder une itération pourront ensuite s'en servir de façon incontrôlée, par exemple pour remédier à des défauts de conception d'un programme. On constate donc qu'une tentative pour faire intégrer un schéma mental de l'itération en terme de fonctionnement (a-priori intéressante puisqu'elle apporte un cadre de plus pour l'interprétation de l'itération), se heurte à des contingences propres à la gestion des capacités de contrôle du dispositif mises à la disposition des élèves.

articulation des différents calculs dans le corps de l'itération. Il semble que même dans le cas où l'itération peut être comprise comme la répétition d'une action contrôlée par un compteur, et où, par conséquent, la condition d'arrêt, et sa coordination avec le corps de l'itération ne posent pas problème, les élèves se trouvent confrontés à une maîtrise insuffisante de l'état des variables qui les empêche d'articuler les actions sur ces variables à l'intérieur du corps de l'itération.

Cette observation va dans le sens ce que nous disions au chapitre 4 (paragraphe 1.4.5) lors de la discussion des propositions de [ARSAC 1980]: la **répétition** (c'est-à-dire l'itération où le nombre de passages dans la boucle est connu) nécessite, tout autant que **l'itération générale**, une interprétation logique de l'état des variables, à partir du moment où interviennent des variables cumulatives. La complexité qui rend nécessaire l'interprétation logique, s'accroît avec le nombre d'éléments interagissant (variables cumulatives, condition d'arrêt..), et non avec la seule introduction d'une (de) condition(s) d'arrêt plus générale(s).

2.1.2 L'aspect "type de données"

Les calculs dans les premières itérations portent sur des données numériques. La conversion binaire-décimal donne lieu au calcul des éléments successifs d'une chaîne de caractères servant à la représentation binaire du nombre : une difficulté propre au traitement des chaînes est le choix et la construction d'un mode de parcours de la chaîne cohérent avec l'algorithme de conversion choisi (ce n'est évidemment pas le même selon que l'on parcourt l'écriture binaire de droite à gauche ou de gauche à droite).

Une chaîne est également parcourue dans l'exercice de comptage des occurrences de la lettre "e" dans une chaîne). Il nous semble que ces problèmes ne sollicitent pas suffisamment la structure de chaîne : en effet, la donnée est de type chaîne, mais le *résultat* est numérique (et plus précisément cardinal) ; d'autre-part, les chaînes apparaissent dans ces problèmes comme des entités formelles plus que comme des données textuelles, ce qui n'incite pas aux différenciations des plans du **S.R.T.** dont nous avons parlé en 1.2.2

2.1.3 L'initiation à l'analyse

Comme on le voit dans le résumé du cours en annexe à ce chapitre (Cours : Du problème à l'algorithme) l'analyse conduit à expliciter un procédé de résolution "manuelle" (c'est à dire par un individu sans machine) sous forme d'une description en langage naturel ; cette description est appelée "*algorithme*" . Ce choix du professeur est conforme au programme officiel de l'option qui préconise, comme seconde étape du "*processus général de résolution de problèmes*", la "*découverte et* (l')*expression d'un procédé de résolution*", la première étape étant l'énoncé du problème, et la troisième le "*codage dans un langage de programmation*".

De façon générale, une analyse adaptée aux problèmes posés aux débutants consiste à repérer les éléments en jeux dans le problème (donnée, résultat, éléments intermédiaires) et à planifier les tâches (découpage et hiérarchisation des modules, choix des représentations...) L'"algorithme" serait la concrétisation de cette analyse sous une forme observable, le manque de recul par rapport à leur activité de résolution ne permettant pas aux élèves d'extérioriser de façon explicite cette activité.[87] Nous

---

[87] Le choix du professeur constitue une interprétation du programme, en ce sens que l'expression du procédé de résolution se fait en langage naturel. Des auteurs comme [DUCRIN 84] ont considéré que le langage naturel ne constitue pas un cadre suffisamment précis pour la concrétisation d'une analyse sous forme d'un algorithme, puisqu'ils introduisent un cadre linguistique plus restrictif (comportant la notion de variable et précisant le type des objets); mais ce cadre linguistique constitue finalement un véritable langage de programmation, non envisagé comme exécutable, et se surajoutant au langage de programmation réellement utilisé, et il n'est pas certain qu'il facilite la tâche des élèves: engagés dans une processus de différentiation au sein de leur S.R.T. entre des plans adaptés directement de leur expérience familière, et des plans adaptés au traitement par un dispositif automatique ; l'introduction d'un langage supplémentaire leur demanderait de construire une différentiation supplémentaire. Un des auteurs partie-prenante du collectif A.DUCRIN prend

avons indiqué ci-dessus (paragraphe 1.2.4) quelles questions pose l'analyse en tant que méthodologie enseignée aux débutants. Nous posons ici deux questions plus précises relatives à la méthodologie exposée par le professeur et au contexte dans lequel l'intervention d'une analyse préalable à la programmation sera observée:

- **la résolution "manuelle"** peut être sans signification pour la programmation du dispositif. Comme nous l'avons vu plus haut, le parcours d'un texte par un individu en vue d'une recherche d'informations est rarement consciemment séquentiel ; l'individu met en oeuvre un procédé de balayage plus ou moins dichotomique et très peu conscient, en rapport avec la connaissance qu'il a du texte, et avec la structuration éventuellement permise par la signification qu'il attribue aux éléments du texte. L'effort d'analyse peut conduire à une explicitation du processus, mais, même explicité, le processus peut rester impossible à traduire en actions exécutables par un dispositif automatique, ou être d'une complexité bien supérieure à ce que permettrait un processus pensé directement pour ce dispositif.
- **le langage naturel** peut se révéler non adapté à l'expression d'algorithmes ; en particulier, les métaphores qu'il rend nécessaires peuvent être prises au pied de la lettre par des débutants ; les structures linguistiques qu'il impose peuvent être trop éloignées des structures propres aux dispositifs automatiques.

2.2 L'épreuve consiste en deux problèmes de "recherche-translation": CLASSE et CRYPTAGE

Le choix des problèmes posés aux élèves dans l'épreuve EPR2 résulte des hypothèses et choix généraux du chapitre **3** et des choix spécifiques énoncés au paragraphe 1 de ce chapitre Notre intention est de demander la construction d'une itération n'accumulant pas les difficultés, donc ne faisant pas intervenir la coordination d'actions dans le corps de boucle. De façon à examiner l'interaction des conceptions relatives aux objets étudiées au chapitre 6, et des conceptions des traitements qui vont apparaître avec l'introduction d'une itération, il est nécessaire que la structure de chaîne soit sollicitée: c'est pourquoi nous pensons important que la donnée et le résulta soient de type chaîne, et que le calcul fasse intervenir un ordinal. Le contexte intuitif des problèmes doit par ailleurs permettre aux élèves de donner une signification aux éléments du problème, de façon à guider leurs anticipations, et à permettre l'utilisation de *plans* issus du contexte familier, l'adaptation de ces plans pour le traitement par le dispositif, et/ou leur remise en cause.

Les deux problèmes consistent donc à programmer le calcul d'une information résultat à partir d'une information donnée, ces deux informations pouvant être représentées par des chaînes. Dans les deux cas également, le traitement proposé utilise une chaîne de caractères (`CLASSE$` pour le problème **CLASSE** , `CODE$` pour le problème **CRYPTAGE**) où les informations sont groupées par couples. Cette chaîne est donc un *texte* au sens que nous avons donné à ce terme au paragraphe 1.2.1 (chaîne de caractères dont un traitement formel peut permettre d'obtenir une information significative), et nous avons vu que la notion de *traitement* sur un texte peut ne pas être immédiate pour un débutant en programmation. La connaissance du premier élément du couple (information donnée) doit permettre d'accéder à l'autre élément (information résultat): un traitement adapté à un dispositif automatique consiste donc dans un premier temps, à calculer *itérativement* la *position* de l'information donnée dans la chaîne, puis à partir de cette position, à opérer une *translation* (c'est-à-dire l'ajout d'un cardinal à cet ordinal) pour obtenir la position de l'information résultat. Le calcul de cette information est ensuite rendu possible par le fait que cette information est de longueur constante (2 pour **CLASSE** , 1 pour **CRYPTAGE**).

Ce type de problèmes, que nous qualifions de "recherche-translation" nous semble donc conforme à nos choix généraux sur les problèmes (chapitre **3**) et aux choix spécifiques de ce chapitre. Les deux problèmes diffèrent, en ce sens que dans le problème **CLASSE**, les éléments du couple sont de nature différente (un nom de

---

d'ailleurs ses distances par rapport à l'application systématique de cette méthode (conclusion de [PAIR 88]).

longueur libre, et une note de deux caractères) alors qu'ils sont de même nature dans **CRYPTAGE**): un caracière, et le caractère qui lui correspond dans un cryptage.

Nous avons fait le choix de proposer le problème **CLASSE** sous forme d'un programme déjà construit, mais où le calcul de certains arguments de fonctions reste à construire ; nous espérons ainsi obtenir une réponse, y compris d'élèves qui ne sauraient pas construire une itération comme solution du problème, et ainsi avoir des informations sur la façon dont ils comprennent l'itération, et dont ils manient à l'intérieur de l'itération les fonctions sur les chaînes.

Dans le problème **CRYPTAGE**, nous demandons la construction intégrale d'un programme, cette construction étant précédée d'une analyse concrétisée par l'expression d'un procédé de résolution en langage naturel : nous voulons ainsi étudier si, pour certains élèves, cette analyse contribue à la construction d'un programme conforme aux contraintes du dispositif, ou, si l'expression en langage naturel et le traitement "manuel" imposent leurs spécificités à l'analyse (conformément aux hypothèses exposée au § 2.1.3 de ce chapitre), et empêchent une expression correcte dans le langage de programmation.

2.2.1 Le problème CLASSE

Exercice 1 :

Dans une classe de 12 élèves, un devoir a été noté de 0 à 20 ; le professeur décide de faire un programme lui permettant, en entrant le nom d'un des élèves, d'obtenir en sortie sa note au devoir.

Son programme utilise une chaîne de caractères rangée dans la variable CLASSE$ et comportant successivement le nom de chaque élèves, suivi de la note qu'il (ou elle) a obtenue au devoir (indiquée avec 2 chiffres)

```
10 LET CLASSE$ ="DUPOND09ALFRED12ARTHUR14LAMBERT11MARIE05VICTOR12
DUPONT16DUVAL14CHARLIE19ARMAND13DUBOIS05DUMONT14"
20 PRINT "Donnez le nom de l'élève "
30 INPUT NOM$ 40 LET LC=LEN(CLASSE$)
50 LET LN=LEN(NOM$)
60 FOR I = 1 TO LC
70 IF NOM$=MID$(CLASSE$,I,...) THEN LET NOTE$
=MID$(CLASSE$,...,...)
80 NEXT I 90 PRINT "La note est ";NOTE$
```

Tu dois compléter les trois emplacement libres (marqués par les "...") de façon à ce que le programme donne bien la note de l'élève.
Ecris dans ta réponse la ligne 70 complétée, et explique.

Les élèves n'ont pas à leur disposition la fonction rendant la position d'une chaîne donnée comme sous-chaîne d'une autre chaîne donnée(fonction POSITION, voir chapitre 4 paragraphe 2.1.3). Le programme doit donc rechercher itérativement la position de début du nom donné (NOM$) comme sous-chaîne de CLASSE$, en comparant successivement cet élément aux sous-chaînes de CLASSE$, de longueur égale, puis, cette position étant trouvée, en déduire la position de début du second élément comme sous-chaîne de CLASSE$. La résolution passe donc par la mise en évidence de cette position comme élément de résolution du problème, et par le calcul, à partir de cette position, de la position de début du second élément. L'itération peut donc se construire à l'aide d'avertisseurs TANT QUE...QT (voir chapitre 4 paragraphe 1.4.4)

```
I := 1
TANT QUE (NOM$ différent de sous-chaîne(CLASSE$,I,...))
    I := I+1
QT
NOTE$  := sous-chaîne(CLASSE$,...,...)
```

ou à l'aide d'avertisseurs REPETER..JUSQU'A

```
I := 0
REPETER
    I := I+1
JUSQU'A (NOM$ = sous-chaîne(CLASSE$,I,...))
```

NOTE$:= sous-chaîne(CLASSE$,...,...)

le calcul de la chaîne NOTE$ étant ensuite direct (non itératif).[88]

Le résumé de l'enseignement reçu par les élèves (annexe 1 au chapitre 6) montre que le professeur a présenté la forme d'itération TANT QUE...dans le cadre des "boucles dont on ne connait pas le nombre de tours". Mais le BASIC utilisé ne permettait pas le codage direct de cette forme ; il aurait donc été nécessaire de coder à l'aide de branchements (GOTO[89]) . Compte-tenu de cette complication, la forme TANT QUE n'a pas été utilisée par les élèves en travaux pratiques (voir paragraphe 2.1.1).

La seule forme d'itération réellement familière aux élèves est la répétition où un compteur parcourt un intervalle donné (codée FOR <compteur> = TO NEXT <compteur>). Cette forme d'itération peut s'employer pour le problème que nous avons proposé, à condition de parcourir systématiquement la chaîne donnée dans toute sa longueur : elle ne permet pas de "sortir de la boucle" lorsque l'information est trouvée. Elle est évidemment moins générale, et seules les contingences propres à la classe où nous avons fait les observations nous ont conduit à l'utiliser dans le programme à compléter. Elle fait intervenir, de notre point de vue, les mêmes difficultés : la compréhension (et la complétion) d'une instruction conditionnelle, l'utilisation d'un index parcourant la chaîne.[90]

Les items à compléter sont :

1 *la longueur de la sous-chaîne à laquelle comparer* NOM$ pour que la condition soit vérifiée lorsque I est égal à la position dans la chaîne CLASSE$ du premier caractère du nom choisi ; la réponse correcte est LN.
2 *la position de début de la sous-chaîne de* CLASSE$, pour que cette sous-chaîne ait comme valeur la note recherchée ; la réponse correcte est I + LN.
3 *la longueur de cette sous-chaîne*, la réponse correcte est 2.

Les réponses pour les items 1. et 3. doivent être faciles pour les élèves, sauf incompréhension profonde du rôle du troisième argument dans la fonction sous-chaîne. Les explications demandées aux élèves pourront apporter des précisions concernant leur compréhension de l'algorithme.

Le deuxième item est un ordinal qui résulte de l'addition (externe) d'un ordinal (I, position de la première lettre du nom dans CLASSE$,), et d'un cardinal (LN, nombre de caractères du nom) ; nous avons repéré dans la première étude les glissements de sens opérés sur le résultat de la fonction "longueur", et nous les avons reliés au double statut "ordinal" et "cardinal" de ce résultat ; le calcul met également en jeu la variable "compteur d'itération" I qui est la seule variable réellement itérative et il se peut que les élèves ne prennent pas en compte cette variable dans le résultat. On attend donc ici la confirmation des conceptions des élèves relatives aux ordinaux, l'articulation de ces conceptions avec leur compréhension de l'itération, en particulier du rôle joué par l'ordinal I (compteur).

2.2.2 Le problème CRYPTAGE

Deux correspondants souhaitent échanger des messages confidentiels.

---

[88] Le cas où un nom entré à la ligne 20 ne serait pas dans la liste, ou serait une sous-chaîne d'un nom existant, n'est pas envisagé. Nous avons voulu en effet, nous limiter à une itération à un seul point de sortie (pour les difficultés supplémentaires introduites par l'itération à plusieurs points de sortie, voir le chapitre 5 paragraphe 1.4.3)

[89] pour plus de précisions sur ces particularités de BASIC, voir la présentation de ce langage au paragraphe 4.1 de l'annexe du chapitre 4. Pour les raisons qui ont conduit le professeur à ne pas faire coder la forme TANT QUE à l'aide de GOTO, voir ci-dessus § 2.1.1.1

[90] Il est possible cependant qu'une forme TANT QUE ou REPETER..JUSQU'A mette en évidence d'une manière différente le compteur d'itération I et fasse intervenir une décomposition différente des calculs. Nous verrons au chapitre **10** le cas d'élèves travaillant avec la forme REPETER..JUSQU'A.

C'est pourquoi ils conviennent du cryptage suivant de leurs message :

- ils conviennent d'une chaîne de caractères comportant les 26 lettres de l'alphabet dans un ordre donné ; on prendra ici la chaîne "azertyuiopqsdfghjklmwxcvbn"

- le cryptage consiste à remplacer chaque lettre du message par la lettre obtenue comme suit : s'il s'agit de la dernière lettre de la chaîne (ici "n"), on la remplace par la première de la chaîne (ici "a"), sinon, on la remplace par la lettre qui la suit dans la chaîne. Ainsi, le message "bonjour" est crypté en "npakpit".

On veut un programme tel que :

- en entrée on donne une lettre de l'alphabet,

- en sortie on obtient la lettre correspondante dans le cryptage.

Ecris l'algorithme et le programme BASIC qui résolvent ce problème.

Comme souligné plus haut, ce problème est isomorphe au problème précédent ; cependant, les objets manipulés sont suffisamment différents pour qu'une simple adaptation de la solution proposée au problème **CLASSE** n'apparaisse pas aux élèves de manière immédiate. Une analyse correcte consiste à partir du résultat (le caractère crypté) ; ce caractère étant dans la chaîne "azert...", on voit alors que pour le calculer, il est nécessaire d'envisager comme élément de solution la position de la lettre donnée dans la chaîne "azert...". La tâche du programmeur se décompose ensuite comme suit :

1 pour trouver cette position construire un parcours itératif de comparaison de la lettre donnée à chaque lettre de la chaîne "azert..." .

2 construire le calcul de la lettre résultat à partir de la position trouvée pour la lettre donnée.

3 coordonner la boucle (structure itérative) et le cas particulier constitué par le dernier caractère de la chaîne.

Dans le problème **CLASSE**, c'était le point 2 qui faisait l'essentiel de la difficulté; ici les élèves ont à construire eux-même le programme, et donc le point 1 constitue une autre difficulté. Deux constructions sont possibles :

- construction 1: on sort de la boucle quand on a trouvé la lettre. On peut envisager une itération TANT QUE, mais, comme nous l'avons indiqué ci-dessus, les élèves doivent coder cette forme avec des instructions de branchement.

```
10 LET CODE$ = "azertyuiopqsdfghjklmwxcvbn"
20 INPUT L$
30 LET I =1
40 IF L$ = MID$(CODE$,I,1) THEN GOTO 70
50       LET I=I+1
60 GOTO 40
70 IF (I<26) THEN LET R$:=MID$(CODE$,I+1,1) ELSE LET
R$ =MID$(CODE$,1,1)
```

- construction 2: on parcourt dans tous les cas toute la chaîne ; la forme FOR..NEXT, la plus familière aux élèves, peut s'employer pour cette construction. Nous donnons à titre d'exemple un programme où le cas particulier de la lettre n (qui étant la dernière de la chaîne, n'est pas suivie de son cryptage) est pris en compte grâce à une astuce permise par cette construction : la variable résultat R$ prend comme valeur initiale le premier caractère (qui constitue le cryptage de n) ; on parcourt la chaîne seulement jusqu'au vingt-cinquième caractère, donc si la lettre n est la donnée, elle n'est pas trouvée, et le résultat garde cette valeur initiale. Cette seconde forme d'itération, bien que moins pertinente, correspond davantage à l'enseignement reçu , et a toutes chances d'être adoptée par les élèves.

```
10 LET CODE$:="azertyuiopqsdfghjklmwxcvbn"
20 INPUT L$
30 LET R$:=MID$(CODE$,1,1)
40 FOR I:=1 TO 25
```

```
50        IF (L$=MID$(CODE$,I,1)) THEN LET
R$=MID$(CODE$,I+1,1)
60 NEXT I
```

La prise en compte du cas particulier constitué par une donnée égale au dernier caractère de la chaîne (la lettre n dans l'exemple) impose de coordonner une alternative et une itération. Cette coordination est plus facile si on envisage l'alternative après l'itération comme dans l'exemple que nous donnons pour la construction 1 ; en effet les contraintes du langage rendent peu commode la présence d'une action complexe comme branche d'une alternative : traiter l'alternative d'abord implique, dans le langage BASIC une instruction de branchement "par dessus" l'itération. Voici un programme de ce type où la partie itération utilise la construction 1.

```
10 LET CODE$ = "azertyuiopqsdfghjklmwxcvbn"
20 INPUT L$
30 IF L$ = MID$(CODE$,26,1) THEN LET I=0 : GOTO 80
40 LET I =1
50 IF L$ = MID$(CODE$,I,1) THEN GOTO 80
60       LET I =I+1
70 GOTO 40
80 LET R$ = MID$(CODE$,I+1,1)
```

De façon à étudier l'influence d'une "analyse" préalable (au sens où cette notion a été présentée par le professeur) sur l'écriture d'un programme effectivement exécutable, nous demandons un "*algorithme*"[91] préalable à l'écriture du programme, dans les termes où ce travail est habituellement demandé par le professeur. Les difficultés essentielles de l'analyse sont la mise en évidence de l'itération, et la coordination entre l'alternative et l'itération ; en effet, l'alternative est clairement indiquée par l'énoncé, et peut "cacher" la nécessité de l'itération, qui, elle, n'apparaît que du fait des contraintes du dispositif. Dans le cas où cette nécessité est comprise, il faut placer l'alternative par rapport à l'itération, et les contraintes du langage conduisent à placer l'alternative à la fin, alors que l'énoncé introduit le cas particulier avant la recherche générale.

3. Analyse des réponses au problème CLASSE

Nous analysons successivement les réponses aux trois items à compléter (présentés ci-dessus au § 2.2.1). Les réponses précises à chacun des items se trouvent dans les annexes à ce chapitre, sous forme de trois tableaux dont la référence est rappelée ci-dessous)

3.1. Longueur de la sous-chaîne à laquelle comparer NOM$ (voir les réponses en Annexe 6 paragraphe 3.1)

La seule réponse erronée est celle de Florence P. (LC) ; l'explication donnée par l'élève permet de penser qu'il s'agit d'une erreur d'écriture. Les explications données par deux élèves (Florence P. et Pierre E.) font mention d'un affichage, et révèlent donc la persistance de confusions entre l'appel de fonctions et l'action de sortie.(*D.Langage.*) D'autres explications (Armelle D. Florence P. , Karine C.) montrent la persistance d'une indifférenciation entre nombre et chaîne (constituante de la difficulté *D.Ord.Card.*). On notera que cette indifférenciation n'empêche pas les élèves de donner une réponse correcte.

---

[91] Nous employons le mot "*algorithme*" au sens où il a été introduit par le professeur en conformité avec le programme officiel de l'option : expression en langage naturel d'un procédé de résolution. Nous avons indiqué en 2.1.3 que cet *algorithme* a pour statut d'extérioriser l'analyse des élèves sous une forme compatible avec leurs capacités.

3.2. position à partir de laquelle rechercher la note comme sous-chaîne de CLASSE$, (voir les réponses en Annexe 6 paragraphe 3.2)

3.2.1 Classement des réponses

- 2 réponses correctes sans explications (Benoît G., Florent R.).
- 2 réponses "I" : (Stéphanie G., Stéphanie Bé.) ; l'explication fournie par Stéphanie G. montre que I n'est pas compris comme compteur parcourant l'ensemble des positions des caractères dans CLASSE$, , mais comme un index repérant chaque élève ; la structure supposée par l'élève est une structure en deux tableaux indexés par I, le premier étant constitué des noms des élèves, l'autre de leurs notes ; les deux tableaux auraient le même identificateur, ce qui peut ne pas être gênant pour l'élève, étant donné le caractère différent des données de chacun des tableaux.
- 10 réponses ne font pas intervenir I. Parmi celles-ci:
    - 7 réponses sont "LN" ou "LN+1"
    - la réponse de Stéphanie Be.[92] peut s'analyser avec les 7 précédentes (oubli d'un "+" ?).
    - la réponse de Alexandra J. ne peut se comprendre que si l'on interprète le "+" et le "-" comme des signes d'opérations portant non sur des nombres, mais sur des chaînes. LC-LN+2 serait donc la chaîne CLASSE$, que l'on aurait privé des caractères jusqu'à NOM$, mais où l'on prendrait 2 caractères.

Parmi ces 10 réponses, on trouve 6 explications selon lesquelles la note se trouve juste après le nom dans la chaîne de caractères. Ces explications montrent que la structuration des données a été bien comprise par ces élèves, et que c'est donc la conception des variables qui est en cause.

3.2.2 Interprétation :

1 la réponse "I" : compte-tenu de l'explication qu'elle fournit, on comprend que l'élève attribue au programme une structure algorithmique influencée par une résolution manuelle : le programme serait une itération où I repérerait les **noms** successifs, au lieu des **caractères** successifs. Dans le **S.R.T.** de l'élève, le *traitement* est bien un processus itératif, mais la *représentation* de l'objet traité n'est pas une chaîne : pour elle le dispositif considère les **noms** et pas les **caractères**.

2 les réponses où I n'intervient pas : Plusieurs hypothèses peuvent être faites :
    - hypothèse 1: les élèves, dans l'anticipation qui les a conduit à la réponse, n'ont en fait envisagé que le cas du premier élève de la liste (ici Dupond) ; les réponses LN + 1 seraient cohérentes avec cette hypothèse ; les réponses LN proviendraient de la conception déjà rencontrée (liée à *D.Ord.Card.*) selon laquelle le dernier caractère d'une chaîne aurait comme position la longueur de la chaîne moins un.
    - hypothèse 2: les caractères jusqu'au premier caractère du nom (non compris pour les réponses LN + 1, compris pour les réponses LN) ont été "supprimés" car "extraits" lors des passages précédents dans la boucle (Conception liée à *D.Sem.Cha.*.) ou, plus subtilement selon l'explication de Karine C., la base de calcul d'une sous-chaîne aurait été positionné à I par les calculs précédents[93].
    - hypothèse 3: LN apparaît aux élèves comme une référence ordinale marquant le dernier caractère du nom (pour les réponses LN + 1), ou son suivant (pour les réponses LN), indépendamment des caractères qui précédent ce nom dans la chaîne ; cette conception est cohérente avec la conception (liée à *D.Ord.Card.*) de la longueur d'une chaîne comme la chaîne

---

[92] Stéphanie Bé. et Stéphanie Be. sont deux élèves différentes.

[93] "(..) *on prend la lettre égale à* I *; à partir de* I, *on prend la longueur du* NOM$". Ayant compté jusqu'à I, il ne serait pas nécessaire de reprendre le comptage depuis le début de la chaîne : on reconnaît l'influence d'un processus "manuel" où un index matériel (par exemple le doigt du même nom !) concrétise la position atteinte..

elle-même, et non comme un un nombre ; car si LN n'est pas un nombre, mais "la chaîne NOM$ toute entière", c'est bien à la suite de cette chaîne qu'il convient de chercher la note.

L'hypothèse 2, bien que cohérente avec des difficultés *D.Sem.Cha.*. rencontrées antérieurement, est, sauf dans le cas de Karine C. , la moins plausible, compte-tenu des explications données par les élèves[94], et de la difficulté qu'ils éprouvent généralement à prendre en compte les passages antérieurs dans la boucle. Une combinaison des hypothèses 1 et 3 paraît la plus plausible. Les conceptions que nous supposons dans chacune des trois hypothèses peuvent en effet fort bien coexister et se renforcer mutuellement : en particulier, les élèves peuvent construire leur réponse à la suite d'une anticipation portant sur le premier nom de la liste (hypothèse 1), puis conformément à l'hypothèse 3, juger que cette réponse est satisfaisante dans le cas général.

Les élèves donnent donc aux ordinaux un statut intermédiaire entre les objets (ici les chaînes) et les quantités (représentées dans le langage comme des nombres) ; ce statut permet d'effectuer des opérations arithmétiques sur ces ordinaux, sans qu'ils cessent d'être considérés comme représentant intrinsèquement l'objet. Alexandra J. est celle qui va le plus loin dans ce domaine (réponse `LC - LN + 2`). On peut considérer aussi que les élèves ne se résignent pas à la perte d'information que constitue la fonction ordinale "`longueur`" ; tout en ne cessant pas d'être un ordinal (pour pouvoir être utilisé dans la construction du second argument de la fonction "`sous-chaîne`"), "`longueur`" doit représenter intrinsèquement la position du dernier caractère de la chaîne concernée (ou la fin de cette chaîne), même si cette chaîne est située à l'intérieur d'une autre chaîne. Cette fausse conception a été analysée au chapitre 6 (présentation au paragraphe 1.3, conclusion au sous-chapitre 6) comme constituante de la difficulté *D.Ord.Card.*; elle apparaît ici dans une grande majorité des réponses.

3.2.3 L'apport d'un problème de "recherche-translation" pour mettre en évidence les conceptions des élèves

Nous avons proposé le problème **CLASSE** et plus généralement les problèmes de "recherche-translation", dans le but de faire apparaître les conceptions spontanées des élèves face à un traitement portant sur un *texte*, et l'influence de leurs conceptions des objets sur la façon dont ils concoivent ce traitement.

En premier lieu, les explications des élèves montrent qu'ils basent sur des traitements familiers leur conception des objets du programme et/ou leur anticipation du fonctionnement de l'itération : index matériel parcourant la chaîne, recherche de noms en noms... L'absence de différenciation de leurs **S.R.T.** entre le niveau des opérations familières et un niveau qui serait adapté à un traitement par un dispositif automatique apparaît donc bien dans les réponses des élèves.

Nous avons montré de plus, comment *D.Ord.Card.* (que nous avons caractérisée dans des problèmes ne faisant pas intervenir l'itération au chapitre précédent) est en jeu dans les réponses des élèves, particulièrement celles qui ne font pas intervenir `I`: dans les conceptions spontanées, les ordinaux ne sont pas distingués des objets qu'ils repèrent, ainsi `LN` (la longueur du nom à rechercher) marque la fin de ce nom, y compris lorsque la recherche s'effectue dans la chaîne `CLASSE$` dont le nom n'est qu'une sous-chaîne. Par contre, le compteur `I` n'a pas de signification, dans la mesure où la recherche n'est pas en fait réellement intégrée comme une itération contrôlée par cette variable. La non-intégration du rôle des variables dans l'itération et la difficulté *D.Ord.Card.* interviennent donc de façon conjuguée dans ces réponses.

3.3. Longueur de la sous-chaîne à extraire pour obtenir `NOTE$` (voir réponses en Annexe 6 paragraphe 3.3)

Les deux seules réponses erronées sont les réponses `LN` (au lieu de 2) de Stéphanie G., Stéphanie Bé. Elles sont cohérentes avec l'interprétation de la structure

[94] A l'exception de Karine C. , les élèves ne mentionnent pas `I` dans leur explication.

que nous avons supposée chez ces élèves (paragraphe 3.2) : la variable I indexerait les noms au lieu des caractères de la chaîne, l'expression MID$(CLASSE$,I,LN) représenterait le nom du Ième élève dans la partie condition de la ligne 70, et la Ième note dans la partie affectation; dans cette interprétation, LN ne serait pas la longueur du mot recherché, mais la longueur de l'élément d'index I (qu'il convient de prendre en entier d'où la réponse LN).

4. Analyse des réponses au problème CRYPTAGE

Nous avons présenté en 2.2.2 une analyse possible du problème **CRYPTAGE** et, à partir de cette analyse, une décomposition de la tâche du programmeur en trois items (recherche de la position du caractère donné, calcul du caractère résultat, traitement du cas particulier). Le problème étant proposé de façon ouverte, l'analyse de l'élève peut différer, et la décomposition proposée ci-dessus pourrait se trouver plus ou moins pertinente. En fait, on le verra ci-dessous, la décomposition reste pertinente, à condition d'admettre que, compte-tenu de l'analyse de l'élève, certains parmi les trois items ne sont pas abordés, et que, d'autre part les items 1 et 2 sont fortement liés : c'est pourquoi nous donnons d'abord une classification des réponses pour chacun des items 1 et 2, de façon à les caractériser séparément (§ 4.1 et 4.2), puis nous cherchons à caractériser leurs liens dans les réponses des élèves (§ 4.3), et de ces liens nous tirons des conclusions. En § 4.4 nous examinons la cohérence des réponses des élèves dans les deux problèmes, de façon à confirmer ces conclusions. En § 4.5 nous analysons le dépouillement du troisième item, en lien avec l'analyse produite par l'élève et le type de calcul du résultat tel qu'il apparaît avec les items 1 et 2 ; nous attendons en effet des réponses à cet item un éclairage sur l'influence des formulations en langage naturel dans l'énoncé et dans l'analyse de l'élève : en effet, l'énoncé du problème en langage naturel donne une place dominante à l'alternative traitant le cas particulier, au détriment de l'itération (voir § 2.2.2)

4.1. Recherche dans la chaîne "azertyuiopqsdfghjklmwxcvbn" du caractère entré par l'utilisateur ; classification des réponses. (voir les réponses en Annexe 6 paragraphe 4.1)

- 4 élèves donnent un algorithme[95] comportant un aspect itératif (présence des termes "*boucle*", ou "*compteur*", ou "*comparer à toutes les lettres*"...). Voici comment ces 4 élèves déduisent un programme de cet algorithme itératif :
    - seul **Benoît G.** écrit un programme correct ; ce programme peut être compris comme la traduction directe de l'algorithme, mais la précision des termes employés dans l'algorithme peut aussi faire penser que cet algorithme est écrit après le programme et en est une traduction.
    - les deux élèves (Stéphanie G., **Stéphanie Bé)** qui mentionnent le mot "compteur" dans leur algorithme, écrivent un programme comportant l'avertisseur de début d'itération, et l'indication des valeurs prises par le compteur (FOR I=1 TO 26) ; mais le corps de l'itération ne comporte pas l'instruction conditionnelle qui permettrait de repérer le rang de la lettre cherchée dans la chaîne ; en cohérence avec la conception des fonctions résultant de difficultés *D.Langage.*, il faut comprendre l'instruction LET MID$=(C$,L$,I+1) qui suit l'avertisseur d'itération comme une *action sur le compteur,* visant à le positionner "juste avant la lettre L$". Il n'y a pas pour ces deux élèves réellement une itération comprise comme une série de passages successifs dans une boucle ; la ligne FOR I=1 TO 26 permet seulement de caractériser la variable I comme *compteur* , la ligne suivante ayant comme fonction de le positionner.
    - Alexandra J. malgré l'indication d'une comparaison "*à toutes les lettres de la chaîne de caractères*" ne parvient pas à traduire son analyse en programme.

[95] Voir note ci-dessus, à propos du terme "*algorithme*"

- Un élève (**Arnaud P.**) écrit un programme (itératif) correct de traitement du cas général ; son analyse mentionne la recherche de la lettre entrée, mais ne précise pas davantage le mode de recherche .
- Deux élèves écrivent une suite de 13 (et non 26) alternatives ; il semble que ces élèves aient compris la transformation comme involutive (ce qu'elle n'est pas)[96], et que l'écriture de 13 alternatives ne leur ait pas semblé insurmontable, (alors que les 26 lignes qui auraient été nécessaires n'ont pas été produites par les élèves qui ont correctement compris la transformation)[97].
- Quatre élèves ne donnent pas d'indication sur la recherche de la lettre donnée dans la chaîne, et produisent un programme "direct" : le résultat est calculé de manière directe, mais erronée, à partir de la lettre donnée.
- Deux élèves ne donnent pas d'indication sur la recherche de la lettre dans la chaîne, et ne produisent pas de programme.
- Une élève donne une très vague indication de recherche et ne produit pas de programme.

En résumé, 4 élèves pensent à une itération, 2 seulement parvenant à une construction itérative correcte. La recherche de la position de la lettre donnée dans la chaîne "azertyuiopqsdfghjklmwxcvbn" est ignorée par 6 élèves sur 14. Nous constatons au paragraphe suivant que ces élèves, influencés par le processus de résolution "manuel" pensent que l'ordinateur peut considérer "directement" la lettre cherchée dans la chaîne, et "inventent" un construction syntaxique appropriée.

4.2. Expression du caractère résultat en fonction du caractère donné ou de sa position : classification des réponses (voir les réponses en Annexe 6 paragraphe 4.2)

- Les deux élèves dont le programme recherche itérativement le rang de l'occurrence de la lettre dans la chaîne calculent correctement la lettre transformée à partir de ce rang (comme sous-chaîne de la chaîne "azert..." commençant en rang+1, et de longueur 1).
- Deux élèves produisent une série de 13 alternatives (nous avons étudié leur réponse ci-dessus).
- Cinq élèves n'écrivent pas de programme.
- Les 6 élèves restant produisent un calcul direct de la lettre résultat à partir de la lettre donnée
    - dans trois cas (Pierre E., Karine C. et Armelle D.), la lettre résultat est calculée comme une sous-chaîne de la chaîne "azert...", de longueur 1, mais avec, comme position initiale *l'addition de la lettre donnée, et du nombre* 1. Il y a une sorte de conversion de type implicite qui transformerait un caractère en sa position dans la chaîne dont on prend la sous-chaîne.
    - l'écriture de Pierre P. se rapproche des trois cas précédents ; néanmoins, la chaîne "azerty..." n'est pas utilisée, et la fonction MID$ a comme premier argument un L entre guillemets, qui doit signifier quelque chose comme "la lettre elle-même" à la différence de L sans guillemets qui aurait un statut plus flou lui permettant d'être utilisée comme ordinal dans l'argument suivant.
    - dans deux cas (Stéphanie G. et Stéphanie Bé.), le résultat est simplement l'addition de la lettre L$, et du nombre 1. Mais on a souligné plus haut que, pour ces élèves, le compteur est "*positionné*" dans la chaîne avant la lettre. Ici, la conversion fonctionne dans les deux sens : un caractère est converti

[96] Une transformation est involutive si la correspondance qu'elle établit entre deux éléments est réciproque ; par exemple en géométrie, les symétries sont involutives.

[97] Il semble qu'il existe, pour des débutants, un seuil en terme de nombre de répétitions, séparant les problèmes où une itération se justifie, et ceux pour lesquels on peut se contenter de répéter explicitement une action (ici le seuil serait compris entre 13 et 26). On comprend qu'une itération où le nombre de répétitions varie selon les données pose une difficulté à ces débutants, un même programme ne pouvant convenir à la fois en dessous et au dessus du seuil.

en sa position par la fonction sous-chaîne; en lui additionnant 1, on trouve une nouvelle position qui est aussitôt convertie en caractère (sans utiliser la fonction sous-chaîne).

4.3 Lien entre la structure algorithmique pensée par l'élève, et sa conception de l'ordinal, position du caractère donné

Nous avons considéré dans les paragraphes précédents, de façon séparée, d'une part le mode d'accès au caractère donné et d'autre part, le calcul du caractère résultat. Cependant, l'analyse cas par cas ci-dessus montre qu'il y a correspondance biunivoque entre les différents modes d'accès au caractère donné et les différents types de calcul du résultat. Pour interpréter cette correspondance, il nous faut examiner comment les éléments du programme s'agencent ; parmi ceux-ci, la position du caractère donné dans la chaîne "azerty..", nous paraît le meilleur point de départ pour cet examen :

- L'élève peut ne pas envisager du tout cette position comme un élément de solution : sa seule possibilité est alors de construire le programme comme un ensemble de cas particuliers (programme en 13 alternatives que nous appelons "*alternatives-involution*").
- Si l'élève assimile la position du caractère donné dans la chaîne au caractère lui-même, le calcul du résultat est constitué d'un appel de la fonction sous-chaîne avec comme second argument la somme du caractère donné et du nombre 1. Il n'y a pas alors, pour l'élève, nécessité d'une itération, ce qui le conduit au programme "*accès-direct*" [98]

  `LET RESULTAT$ = MID$(ALPHABET$, DONNEE$ + 1, 1)`
- Si, par contre, l'élève donne à cette position un statut **d'ordinal**, il lui faut construire une relation entre cet ordinal et le caractère donné ; pour quatre élèves, ce mode de relation est une itération, et l'ordinal apparaît comme une valeur prise à un moment donné par le compteur d'itération ;
- Un calcul correct du résultat suppose alors la construction d'une alternative, dont la "condition" est réalisée précisément quand le compteur prend comme valeur cette position.
- Mais cette alternative peut être difficile à construire pour les élèves et/ou le mécanisme de valuation du compteur peut ne pas être compris ; le résultat est un programme curieux (que nous appelons "*Stéphanies*", les deux élèves qui l'écrivent portant ce prénom) :

programme "STEPHANIES" (C$ est la chaîne "azertyui..", L$ est le caractère donné).	`FOR I=1 TO 26`
	`LET MID$=(C$,L$,I+1)`
	`Print L$+1`

Ce programme commence comme une itération ; suivent deux instructions témoignant d'une conception naïve du langage ; la première pourrait avoir pour signification "positionner le compteur à la position suivant la lettre donnée" ; la seconde contient la confusion de type qui se traduit par l'ajout d'un nombre à un caractère. Dans le **S.R.T.** des élèves du programme "*Stéphanies*", les avertisseurs

---

[98] Selon [MORVAN 81], l'"*accès*" est «*l'opération permettant d'obtenir le contenu d'un enregistrement physique ou d'un mot mémoire*». L'*accès séquentiel* «*ne permet d'atteindre l'enregistrement d'ordre n qu'après obtention des n-1 enregistrements précédents*». C'est le type d'accès compatible avec la structuration proposée dans l'énoncé. Par contre, dans leur solution, les élèves pensent à l'*accès direct* que [MORVAN 81] définit comme le mode permettant «*d'atteindre chaque enregistrement indépendamment des autres*». On peut aussi caractériser cette divergence entre le mode d'accès permis par la structuration des données et celui auquel pensent les élèves, en se référant à la notion d'*adressage*, que [MORVAN 81] définit comme la fonction «permettant de sélectionner un élément parmi un ensemble d'éléments du même type par calcul d'une adresse». En effet, l'index i est ici l'*adresse* si l'on considère la chaîne ALPHABET$ comme la partie de la mémoire où sont rangées les informations. Il en résulte que la structuration des données de l'énoncé est compatible avec l'adressage indexé classique, alors que les élèves pensent à un adressage *par le contenu* (ou associatif), dans lequel l'organe de calcul accède à l'information non par la connaissance de son adresse, mais par celle de sa valeur. De façon plus générale, la conduite des élèves résulte d'une indifférenciation entre l'adresse et la valeur.

d'itération donnent à une variable le statut de compteur ; ce **S.R.T.** n'intègre pas le mécanisme de valuation de ce compteur ni la nécessité que le résultat soit calculé à partir de ce compteur : on le "positionne" (`LET MID$=(C$,L$,I+1)`), puis on affiche le résultat sans référence à ce compteur (`Print L$+1`)

- 2 élèves n'entrent pas dans les 3 catégories énoncées ci-dessus : l'une envisage une itération (*"la comparer à toutes les lettres"*), mais ne prend pas en compte la position du caractère donné : on peut penser qu'ainsi, la construction de l'itération est dans une impasse, puisque l'élève ne donne pas de programme. L'autre élève donne une indication vague concernant la position du caractère donné (*"on cherche la place..."*) mais n'envisage pas l'itération qui permettrait de calculer cette position ; elle ne donne pas non plus de programme
- 2 élèves ne produisent pas d'éléments significatifs (l'analyse paraphrase l'énoncé et il n'y a pas de programme).

L'analyse ci-dessus conduit à trois schémas de résolution (plus deux si l'on prend en compte les élèves qui n'écrivent pas de programme) ; ces schémas sont compatibles avec les réponses des élèves, mais le raisonnement réel qui a conduit à ces réponses peut fort bien ne pas respecter ces schémas, par exemple, dans une démarche plus ascendante, l'élève peut avoir l'intuition d'une itération, ce qui implique, dans la forme d'itération qui est familière dans cette classe, la présence d'un "compteur", entité numérique à laquelle il donne une signification en la reliant à la position du caractère donné. Les schémas permettent cependant de décrire un agencement entre les éléments de résolution qui nous paraît ne pas dépendre de l'ordre dans lequel les élèves les font intervenir dans leur raisonnement. Nous résumons cet agencement dans le tableau suivant (chacun des schémas de résolution est décrit dans une ligne du tableau)

Tableau n°1 : les schémas de résolution dans le problème CRYPTAGE

position du caractère donné	structure algorithmique	programme	nb élèves
pas de mention de la position	13 alternatives	*alternatives-involution*	2
assimilation caractère-position	directe	*accès direct*	4
comme ordinal	itération avec alternative	programme correct	2
	itération sans alternative	*programme "Stéphanies"*	2
pas de mention	*comparer à tous les caractères*	pas de programme	1
voir la place	pas de recherche	pas de programme	1
pas d'analyse		pas de programme	2

L'analyse ci-dessus et le tableau qui la résume montrent que la façon dont les élèves considèrent la **position du caractère donné** est liée fortement à la façon dont ils se représentent le **traitement** par le dispositif :

- dans le cas du programme *accès direct*, les élèves pensent que, comme argument de la fonction `MID$`, cette position peut-être assimilée au caractère lui-même ; il n'y a pas lieu de construire un algorithme de recherche de cette position.
- dans le cas des programmes corrects et du programme "*Stéphanies*", les élèves aperçoivent la nécessité qu'un **ordinal** prenne à un moment donné comme valeur cette position et construisent un parcours de la chaîne `"azertyu..."`.

Une meilleure conception de la structuration des données (ici la conception d'un *index* comme une entité numérique précise) est donc à relier à une différenciation au sein du **S.R.T.** entre un traitement spécifique au dispositif et le traitement familier. Nous retrouvons donc dans cette situation expérimentale, **l'interdépendance entre la structuration des données et la structure algorithmique**, que nous avons pointée comme une question importante dans l'étude épistémologique du chapitre 1 (§ 5.4), et concernant le **S.R.T.** du sujet, nous confirmons l'hypothèse (§ 1.2.3 de ce

chapitre) d'une **intervention conjointe d'un plan relatif au traitement, et d'un plan relatif aux données** : cette intervention se traduit par un programme (ou une analyse) ***cohérent avec chacun des deux plans.***

Avec le programme "*Stéphanies*", nous nous trouvons dans un cas où le plan du traitement est itératif (donc relativement différencié du traitement "naïf"), où le plan relatif aux données fait apparaître un ordinal, mais où le programme est erroné. Ce programme peut être mis en relation avec des difficultés classiquement repérées dans l'itération.(construction d'une alternative, utilisation de la variable `compteur`) tout en étant marqué par l'utilisation naïve de l'affectation et de la notation fonctionnelle que nous avons désignée au chapitre 6 sous le nom de `D.Langage.` ; le programme "*Stéphanies*" est donc aussi le résultat *cohérent* de **l'intervention conjointe d'un plan relatif au traitement** (caractérisé par une compréhension naïve de l'itération), **et d'un plan relatif aux chaînes** (caractérisé par l'utilisation naïve du langage).

Le schéma ci-dessous résume l'intervention conjointe des différents conceptions dans un programme (verticalement, les plans relatifs aux données, horizontalement les plans relatifs aux traitements).

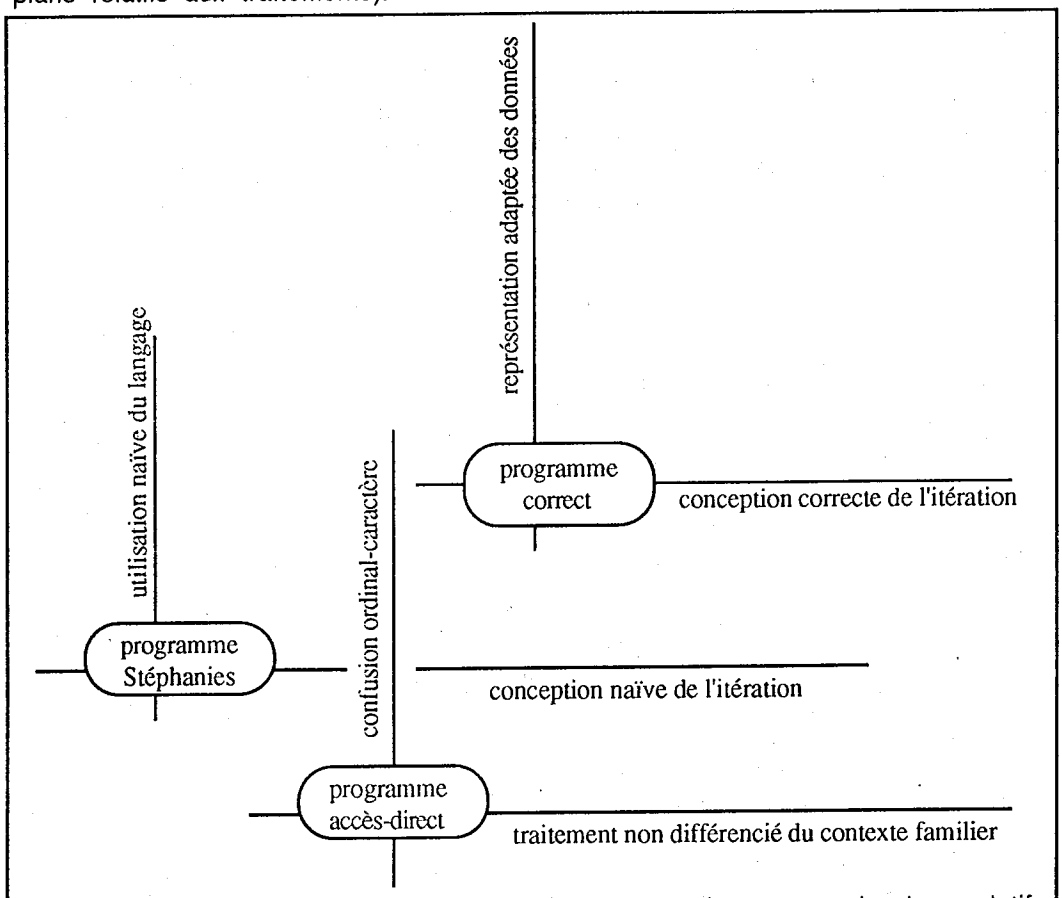


Schéma : l'intervention conjointe de plans relatifs aux traitements et de plans relatifs aux données dans chacun des programmes.

Nous ne pouvons savoir, à partir des réponses des. élèves, si, dans cette intervention, un des plans a déterminé l'autre: nous avons en effet indiqué que les schémas de résolution sur lesquels nous nous appuyons permettent de situer l'agencement des éléments du programme, mais non l'ordre dans lequel les élèves les font intervenir dans leur raisonnement.

Indiquons cependant une hypothèse dans le domaine de la genèse des acquisitions : la conception de la position pourrait jouer un rôle important dans l'amélioration du **S.R.T** ; l'addition d'un caractère et d'un nombre étant une opération impossible pour le dispositif, l'apprenti programmeur devra distinguer le caractère de sa position, et donc rééquilibrer ses plans relatifs aux objets et par conséquent la façon dont il considère le *traitement* , puisque cette position n'étant pas donnée, et apparaissant nettement

comme numérique, il est alors nécessaire de la **calculer.**

En se référant au schéma ci-dessus, un déplacement vers la droite à partir du programme "accès direct", en restant sur le plan horizontal nous positionne sur le plan vertical de la conception des données adaptée, mais crée un déséquilibre (il n'y a pas de programme à l'intersection de ces conceptions) et donc la nécessité de se déplacer horizontalement.

4.4 Lien entre le mode de recherche de la lettre donnée dans le problème CRYPTAGE et la présence du compteur (I) dans le calcul de la position du premier caractère de la note dans le problème CLASSE

Il y aurait donc des élèves qui considèrent que la recherche d'informations dans une chaîne impose une itération, et font intervenir le compteur `I` comme un index sur la chaîne et d'autres qui pensent que l'accès à l'information est "direct", et négligent ce compteur. Si cette interprétation est correcte, on doit trouver une corrélation entre la réponse au problème 1 (en particulier la position à partir de laquelle rechercher la note) et la présence d'une recherche itérative de la lettre donnée, dans le problème 2 que nous étudions ici. Nous étudions cette corrélation dans le tableau suivant.:

Tableau n° 2

problème CRYPTAGE	problème CLASSE	
	compteur dans la position de la note	pas de compteur dans la position de la note
indication d'une itération dans l'analyse ou le programme	3	2
pas de recherche		7
recherche par 13 alternatives	1	1

Interprétation

Pour dix élèves sur quatorze, le tableau fait apparaître une corrélation certaine entre les réponses au problème 1 et au problème 2 :

- les 3 élèves de la case en haut à gauche prennent en compte la variable `I` dans le problème 1, et construisent une itération dans le problème 2 ; on peut penser qu'ils comprennent le programme du problème 1 comme itératif, et que d'emblée, ils conçoivent la solution du problème 2 comme itérative.
- les 7 élèves de la seconde case de la deuxième ligne conçoivent la solution du problème 2 comme "directe" (sans itération) ; ils ne prennent pas en compte la variable `I` dans leur réponse au problème 1 ; une explication à cela serait qu'ils ne comprennent pas le programme à compléter au problème 1 comme itératif.

Le problème **CLASSE** est une tâche de complétion, le problème **CRYPTAGE** une tâche de création et ces problèmes diffèrent par leur contexte intuitif. De la comparaison des conduites, nous déduisons une solidité des conceptions au moins chez certains élèves, puisqu'elles apparaissent semblables dans les deux tâches et les deux contextes.

4.5 Prise en compte du cas particulier constitué par la lettre "n" (voir réponses en Annexe 6 paragraphe 4.3).

4.5.1 Classification des réponses

- six élèves prennent en compte le cas particulier ; parmi eux, seul **Benoît G.** l'inclut dans un programme itératif, mais ne le coordonne pas correctement avec le reste du corps de boucle.
- les six élèves restant ne prennent pas en compte le cas particulier ; chez quatre d'entre-eux, une prise en compte de l'aspect itératif de la recherche de la position du caractère donné a été repérée ci-dessus.
- pour deux élèves, il n'y a pas nécessité de prise en compte particulière (du fait de la structure en 13 alternatives, ce qui correspond au programme "alternatives-involution")

4.5.2 Lien entre le mode de recherche de la lettre donnée et la prise en compte du cas particulier

Tableau n°3	prise en compte du cas particulier	pas de prise en compte du cas particulier
Indication d'une itération dans l'analyse ou le programme	1	4
pas d'itération	5	2
recherche par 13 alternatives	2 la question du cas particulier ne se pose pas	

Interprétation :

Le tableau fait apparaître deux catégories significatives d'élèves :

- une catégorie constituée de 4 élèves qui envisagent ou réalisent un traitement itératif pour le cas général, mais ne prennent pas en compte le cas particulier
- une catégorie constituée de 5 élèves qui prennent en compte le cas particulier, mais se représentent le traitement général comme non-itératif (4 d'entre-eux réalisent en fait le programme que nous avons appelé "*accès-direct*" au paragraphe 4.3).

Les deux catégories d'élèves nous semblent correspondre à deux façons d'analyser le problème.

- La première catégorie considérerait que le traitement itératif constitue "le gros morceau" du travail, et qu'il faut l'envisager prioritairement, ce qui est pertinent puisque cette partie pose la question des données (introduction d'une variable). Pourquoi ne passent-ils pas à la seconde étape? Peut-on penser qu'ayant traité le cas général, ils estiment que le travail est terminé?[99] Nous reverrons cette question dans l'analyse d'un entretien. Cette conduite prouve en tout cas que ces élèves considèrent effectivement le cas d'une entrée "n" comme un cas particulier.
- Pour les élèves qui ne conçoivent pas le traitement général comme itératif, la hiérarchie des tâches est différente puisque, dans leur **S.R.T.** , ce traitement est moins volumineux (en travail de conception comme en lignes de programmes) et n'implique pas la création de variables. Le traitement général et le traitement particulier étant d'égale importance, et leur coordination ne posant pas problème, ils les traitent ensemble.

La prise en compte du cas particulier paraît donc liée étroitement à la conception que se font les élèves du traitement, qui lui même est en lien avec la conception des données (§ 4.3). Nous avons fait l'hypothèse que la formulation de l'énoncé, et la conception de l'analyse présentée par le professeur en cours (expression en langage naturel d'un procédé de résolution "manuel") pouvait jouer un rôle en mettant l'accent sur la structure alternative. Ces formulations en langage naturel ont pu jouer un rôle dans l'analyse des élèves qui écrivent le programme "*accès-direct*", mais nous avons vu que ce type de programme est déjà fortement déterminé par la conjonction de conceptions du traitement et des données (§ 4.3).

Nous verrons au chapitre 9 que lors d'une épreuve passée dans les mêmes conditions, avec une situation similaire, mais dans une modalité où le caractère "a" est répété à la fin de la chaîne "azerty..." de façon que l'entrée "n" ne soit plus un cas particulier, on obtient de certains élèves des programmes du type "*accès-direct*". Par contre, les élèves qui conçoivent le traitement du cas général comme itératif ne se

[99] Des observations montrent que des élèves débutants peuvent se contenter d'un programme effectuant la tâche demandée dans les cas les plus courants et ne pas considérer comme faux un programme qui n'effectue pas la tâche dans des cas qu'ils jugent marginaux [ROGALSKI 88].

laissent pas influencer par la formulation de l'énoncé (au point qu'ils oublient de traiter le cas particulier), bien qu'on rencontre parmi eux des sujets dont le **S.R.T.** est loin d'être adéquat.

Au total, l'analyse, au sens d'expression en langage naturel d'un procédé de résolution manuel, semble avoir joué un rôle assez neutre, comme le montrent les tableaux suivants.

4.6 Comparaison entre l'expression en langage naturel (*algorithme*), et le programme.

La recherche du caractère donné dans la chaîne "azert.."

Tableau n°4	programme itératif	programme non-itératif	pas de programme
algorithme itératif	3		1
recherche non explicitement itérative	1	2	1
pas de recherche		4	2

La prise en compte de l'entrée "n"

Tableau n°5	prise en compte dans le programme	non prise en compte dans le programme	pas de programme
prise en compte dans l'algorithme	5		1
non prise en compte dans l'algorithme		6	2

Interprétation

On peut penser à une influence de l'analyse si, par exemple, l'algorithme présente des éléments déduits de l'énoncé, marquant une progression vers un traitement compatible avec le dispositif, mais demandant une adaptation dans le programme : l'algorithme constitue alors une étape de la résolution, en tant que description d'un traitement compatible avec un dispositif général ; le programme est alors une adaptation de ce traitement au dispositif particulier sur lequel les élèves programment. C'est le cas d'un seul élève, **Arnaud P.**, dont l'algorithme comporte la mention "*La lettre du message non codé est recherchée dans la chaîne de caractères*", son programme étant une utilisation correcte de la boucle FOR..NEXT. La notion de recherche du caractère à crypter n'est pas dans l'énoncé, et elle marque la prise en compte des contraintes d'un dispositif général. Deux autres élèves font également apparaître une notion assez vague de recherche ("voir sa place", "comparer à tous les caractères", mais on peut penser qu'elles échouent à adapter cette notion au dispositif sur lequel elles programment, puisqu'elles n'écrivent pas de programme.[100]

Chez les 11 autres élèves l'analyse demandée sous-forme d'un algorithme en langage naturel apparaît en fait très liée au programme, puisque les éléments présents dans l'algorithme le sont également dans le programme, et sous des formes très proches: soit les élèves pensent que la formulation de l'énoncé est directement compatible avec le traitement par le dispositif, et dans ce cas leur algorithme paraphrase l'énoncé, et leur programme en constitue une adaptation syntaxique ; soit ils pensent que le traitement de l'énoncé ne peut être conforme à l'énoncé, et ils conçoivent de façon indifférenciée l'algorithme et le programme.

[100] Ces trois élèves occupent la seconde ligne du tableau 4 ci-dessus.

5. Confirmation de l'interaction Objets-Traitements (analyse d'un entretien)

Nous avons fait passer un entretien, c'est-à-dire, comme au chapitre précédent, une séance de résolution en présence de l'ordinateur, où les élèves travaillent sous notre direction à partir de leurs réponses à l'épreuve écrite. L'entretien a été enregistré et on en trouvera le dépouillement dans le volume Annexes (Annexe 7 paragraphe 5). Au chapitre précédent, ne disposant pas encore d'un cadre d'analyse permettant d'interpréter directement les réponses à l'épreuve écrite (EPR1) nous avons dû nous appuyer sur plusieurs entretiens avec des élèves de niveaux différents. Ici, nous avons pu, grâce aux acquis du chapitre précédent, interpréter plus facilement les réponses à EPR2. L'unique entretien que nous analysons ici vient en fait confirmer les conclusions du dépouillement, c'est pourquoi, contrairement à la première épreuve, nous n'avons pas jugé utile de procéder à d'autres entretiens.

5.1 Les conceptions des élèves

Concernant la conception que se font les élèves des fonctions ordinales, et particulièrement de la fonction "longueur", Florence P. peut reconnaître que `LN` vaut 7, et à la suite, déclarer que `LN + 2` vaut «`LAMBERT plus 2 caractères`». L'entretien confirme également la conception erronée de la variable `I` comme index sur les noms, y compris après que la solution ait été mise en évidence : l'hypothèse de récurrence (ou invariant d'itération) implicite pour les élèves est "`I` représente la position du nom que l'on compare à `NOM$`", mais la signification de `I` reste imprécise.

- En tant que "compteur", la variable `I` est parfois comprise comme indexant les noms dans la chaîne. Pour Florence P. et Alexandra J., c'est le "nom de chaque élève" , et à la question «*si je cherche* `ALFRED`, *par exemple,* `I` *va être égal à combien?*» Armelle D. répond : «*Alfred est en deuxième position*» .
- Par contre, en tant que constitutif d'une expression argument (ordinal) de la fonction "`sous-chaîne`" , la variable `I` représente "ce qu'il y avait avant".

Le fonctionnement de la boucle `FOR...NEXT` et le compteur lui-même, restent donc mal interprétés par les élèves ; la conception naïve (le langage est utilisé comme dans la communication habituelle) domine au détriment d'une compréhension précise du fonctionnement du dispositif.

5.2 Le problème CRYPTAGE et le traitement du cas particulier

Pour le problème **CRYPTAGE**, la nécessité d'une itération s'impose d'emblée aux élèves, alors qu'aucune des réponses à l'épreuve écrite des trois élèves concernées par l'entretien ne comporte de programme itératif ; il s'est produit une maturation dans laquelle la résolution du problème **CLASSE** joue un rôle (ce qui confirme que la solution proposée dans l'énoncé de ce problème n'a pas, au cours de l'épreuve papier, été comprise comme itérative) Le problème **CRYPTAGE** est reconnu isomorphe au problème **CLASSE**, et la résolution est calquée sur la solution proposée dans l'énoncé pour le problème **CLASSE** (alors qu'il y aurait une simplification résultant de ce que la longueur de l'élément cherché est 1). Cette conduite confirme, pour ces trois élèves, le fait qu'une évolution de la conception des données entraîne une rééquilibration dans le domaine du traitement (hypothèse que nous avions avancé en 4.3).

Nous nous étions interrogé en 4.4 sur la raison pour laquelle les élèves qui construisent un traitement itératif, dans leur presque totalité, ne traitent pas le cas particulier. Deux des trois élèves concernées par l'entretien prenaient en compte le cas particulier lors de l'épreuve sur papier, la troisième n'abordant, en fait pratiquement pas la résolution de ce problème. Elles entreprennent lors de l'entretien, le traitement itératif du cas général, sans se préoccuper du cas particulier, ce qui confirme notre analyse en 4.6 selon laquelle, c'est la conception d'une représentation et d'un traitement par le dispositif qui détermine l'intérêt porté au cas particulier, et non la structure linguistique de l'énoncé. Lors des essais, Alexandra J. déclare : «*je vais faire* `n` *pour voir si ça fait* `a`», et Florence P. semble déçue de ne pas obtenir le résultat : «*Oh, ça recommence...*».

Il semble bien que dans l'anticipation des élèves, un traitement conçu pour le cas général puisse avoir la capacité de donner le résultat correct dans le cas particulier. Le "feed-back" de l'exécution remet vite en cause cette anticipation, Florence P. imagine facilement un traitement alternatif du cas particulier, par contre, elle éprouve une grande difficulté à coordonner cette alternative et le traitement itératif du cas général. La difficulté essentielle tient à l'indifférenciation entre l'affectation qui donne sa valeur à la variable résultat et l'affichage de ce résultat. La coordination entre deux plans de traitement paraît donc nécessiter l'immersion de ces plans dans un **S.R.T.** général où le langage est bien intégré, alors qu'un traitement dans un seul plan peut fort bien être réalisé malgré une intégration rudimentaire du langage.

6. Conclusions

Le but principal de ce chapitre est d'examiner l'intervention de plans du **S.R.T.** de l'élève, relatifs aux traitements, en conjonction avec des plans relatifs au données étudiés au chapitre précédent. Nous avons fait le choix de l'itération qui est en cours d'acquisition chez les élèves et d'une classe de problème: les problèmes de "recherche-translation". Dans ces problèmes, une chaîne constante est constituée de la concaténation d'informations données et d'informations résultats; il s'agit de construire une itération permettant de calculer la position d'une information comme sous-chaîne de la chaîne constante, d'obtenir ensuite l'information résultat en opérant une "translation" dans cette chaîne. Les problèmes de cette classe permettent de faire intervenir l'itération dans une situation qui n'accumulent pas les difficultés propres à cette structure algorithmique, tout en sollicitant fortement la structure de chaîne. D'autre part, le contexte du problème permet aux élèves d'attribuer un sens aux objets en jeu, et d'utiliser des plans issus du traitement familier.

L'analyse des réponses au problème **CLASSE** montre que les élèves donnent aux ordinaux un statut intermédiaire entre les objets (ici les chaînes) et les quantités (représentées dans le langage comme des nombres); ce statut permet d'effectuer des opérations arithmétiques sur ces ordinaux, sans qu'ils cessent d'être considérés comme représentant intrinsèquement l'objet. Nous avons analysé au chapitre 6 cette conception comme partie intégrante d'un plan relatif aux données et comme constituante de la difficulté *D.Ord.Card.*. Elle apparaît sous plusieurs formes dans une majorité des réponses. La réponse au premier item, exacte dans presque toutes les réponses, n'entre pas, par contre, en contradiction avec cette conception.

La non-intégration du rôle des variables dans l'itération (constituante d'un plan relatif au traitement) intervient conjointement avec cette conception. Certaines explications des élèves montrent qu'ils basent sur des traitements familiers leur conception des objets du programme et/ou leur anticipation du fonctionnement de l'itération. Notre hypothèse concernant la non-différenciation, entre le **S.R.T.** correspondant au contexte familier, et celui correspondant à un dispositif automatique se confirme chez une majorité d'élèves.

Dans les réponses au problème **CRYPTAGE**, nous avons mis en évidence 4 classes de réponses: dans l'une, les élèves envisagent le programme comme une série de 13 cas particuliers, sans utiliser la structure de données pourtant proposée dans l'énoncé. Les 3 autres classes nous semblent confirmer nos hypothèses : dans les programmes de la classe "*accès-direct*" la confusion ordinal-caractère (*D.Ord.Card*) est présente en conjonction avec une forme de traitement directement adaptée du traitement dans le contexte familier ; le programme "*Stéphanies*", résulte de la conception naïve du langage (*D.Langage*) et d'une intégration insuffisante des éléments de l'itération ; le traitement correct, que produisent deux élèves seulement, suppose à la fois une conception des données compatible avec l'itération, et une bonne maîtrise de cette structure.

Ces réponses s'analysent comme des points d'équilibre résultant d'une conjonction d'une conception des données et d'une conception de l'itération. Les réponses erronées témoignent de *plans* issus d'un **S.R.T.** correspondant à un traitement des textes dans le contexte familier, mais insuffisamment adaptés au

traitement par le dispositif. Pour dix élèves sur quatorze, les réponses au problème 1 et au problème 2 sont très cohérentes. Nous en déduisons une solidité des conceptions puisqu'elles apparaissent semblables chez la majorité des élèves, dans deux tâches et deux contextes assez différents.

Les résultats obtenus confirment également que la classe de problèmes "recherche-translation dans une chaîne" à laquelle appartiennent les deux problèmes proposés dans l'épreuve mettent bien en évidence cette interaction entre objets et traitements et cette influence du contexte familier.

L'analyse d'un entretien indique que chez les élèves concernés, la conception des ordinaux liée à *D.Ord.Card.* (relative aux données) semble jouer un rôle déterminant. Un travail conduisant à une différenciation dans un plan relatif aux données permet aux élèves d'envisager ensuite spontanément un traitement itératif du problème **CRYPTAGE**, et donc d'opérer une différenciation dans une plan relatif au traitement. Nous avons donc un exemple où un "déséquilibre" provoqué par la remise d'une conception relative aux objets conduit à une rééquilibration dans la conception du traitement.

Nous avons constaté également que la nécessité de coordonner deux plans relatifs au traitement, l'une correspondant à un traitement alternatif, l'autre à un traitement itératif fait ressortir des difficultés dues à la non-intégration du langage (*D.Langage*). Cette coordination paraît donc nécessiter l'immersion de ces conceptions dans un **S.R.T.** où les éléments du langage relatifs aux objets sont bien intégrés, alors qu'un traitement dans un seul plan peut fort bien être réalisé malgré une intégration rudimentaire du langage; la nécessité d'une intégration suffisante des propriétés des objets, pour permettre la coordination de plans relatifs aux traitements nous paraît une nouvelle illustration des relations au sein du **S.R.T.** entre les deux types de plans, que nous tentons de mettre en évidence dans ce chapitre.

Nous avons voulu également dans ce chapitre examiner le rôle joué par les formulations en langage naturel de l'énoncé, et de l'analyse préalable à la programmation. Nous avons fait l'hypothèse selon laquelle la formulation de l'énoncé du second problème, et la méthode d'analyse présentée par le professeur renforceraient les fausses conceptions en mettant l'accent sur la structure alternative correspondant au cas particulier, au détriment du traitement itératif du cas général. On peut penser à un processus d'analyse correspondant effectivement à un dispositif plus général que celui pour lequel ils programment dans le cas de trois élèves, dont un seul parvient à déduire un programme de cette analyse. Les autres élèves conçoivent manifestement leur solution dans un **S.R.T.** qu'ils estiment directement adapté au dispositif : une partie des élèves pense que la structure présentée dans l'énoncé est directement compatible avec le dispositif, et leur programme en est une simple transposition syntaxique ; les autres opèrent une différenciation plus ou moins approfondie, et ils ne sont plus influencés par la formulation de l'énoncé, au point que, dans la plupart des cas, ils oublient de traiter le cas particulier: il n'y a pas d'évolution du plan utilisé par l'élève, de l'analyse en langage naturel au programme. Les formulations en langage naturel de l'énoncé, ou de l'analyse, bien que mettant l'accent sur la structure alternative, semblent ne pas avoir une influence déterminante sur les conduites des élèves.

Chapitre 8 Etude quantitative

Une épreuve visant à faire le point sur les acquisitions des élèves dans le domaine des chaînes de caractères en fin d'année de seconde "option informatique". Passation dans trois classes

A la suite de l'étude à dominante clinique rapportée au chapitre **6** et **7**, nous avons fait passer dans trois classes de seconde option informatique une épreuve sur papier. Dans la suite de la thèse, nous appelons EPR3 cette épreuve. Pour mémoire, EPR1 et EPR2 désignent respectivement les épreuves analysées aux chapitres **6** et **7**.

Dans ce chapitre, nous indiquons la place que tient cette épreuve dans notre thèse (paragraphe 1), et les choix que nous avons dû faire quant à la forme de l'énoncé et aux éléments observés (paragraphe 2), puis nous présentons l'énoncé de l'épreuve (paragraphe 3). On trouvera ensuite une analyse des réponses des élèves (paragraphe 4) suivie de l'analyse des relations entre la réussite à cette épreuve, le niveau scolaire général de l'élève, et les activités menées dans la classe (paragraphe 5). Le paragraphe 6 tire les conclusions du chapitre.

Une annexe (annexe 8) à ce chapitre donne une variante de l'énoncé (variante passée dans une des classes) et la liste complète des réponses des élèves.

1. Vérifier la généralité des résultats du chapitre 6, opérer des comparaisons

Nous avons, au cours des chapitres précédents indiqué les difficultés rencontrées par les élèves d'une classe de seconde option informatique au cours de la résolution de problèmes simples impliquant la mise en oeuvre de fonctions sur les chaînes de caractères. Rappelons que nous avons distingué en premier lieu des difficultés révélatrices de l'inadaptation de plans relatifs aux objets :

- les difficultés *D.Langage*, liées aux rapports entre les objets de type chaîne et les éléments généraux de la programmation (en particulier une difficulté à donner un sens adéquat aux écritures fonctionnelles, et à l'affectation quant elles portent sur ces objets),
- les difficultés *D.Sem.Cha.* liées à une incompréhension de la logique interne de la structure (arguments de la fonction `sous-chaîne`, effet de la `concaténation` sur le dispositif..)
- les difficultés *D.Ord.Card.* dans le calcul des arguments numériques des fonctions, en particulier celles qui révèlent une incapacité à donner un sens adéquat aux opérations sur les ordinaux.
- la difficultés *D.Type* qui conduit l'élève à faire le choix d'un type de donnée en fonction de son apparence externe et non du jeu de fonction qu'il fait opérer sur ces données.

Dans un second temps, nous avions mis en évidence l'**interaction** entre les plans relatifs aux objets, et les plans relatifs aux traitements au sein des **S.R.T.** que les élèves mettent en oeuvre pour la résolution de problèmes faisant appel à l'itération sur une chaîne de caractères..

L'étude ayant conduit à ces conclusions portait sur un petit nombre d'élèves d'une même classe. Cette étude à caractère clinique était nécessaire pour construire un cadre d'analyse pour l'interprétation des difficultés observées. En contrepartie, ces difficultés pourraient résulter de facteurs spécifiques à cette classe ou à ces élèves. Le premier objectif de cette épreuve sur papier s'adressant à plusieurs classes de seconde option informatique est donc de vérifier le caractère général au chapitre précédent. Un second objectif est d'opérer des comparaisons entre classes, et entre groupes d'élèves constitués selon un critère général de réussite dans les disciplines scientifiques.

2. Questions ouvertes et questions fermées

Comme dans les études précédentes, cette épreuve est constituée de problèmes ou exercices présentant une situation supposée connue des élèves, et demandant la rédaction d'un programme dans le langage de programmation utilisé par la classe, éventuellement accompagné d'un commentaire d'explication. Ces problèmes ou exercices à traiter sur papier peuvent être posés de différentes façons :

- de façon "*ouverte*", le problème est posé en termes du langage courant, un début de programme peut amorcer une codification des données, mais le choix des variables et la construction de la structure algorithmique sont à la charge de l'élève.
- de façon "*fermée*", le problème est posé comme ci-dessus, mais les variables et la structure algorithmique sont indiquées sous la forme d'un programme comportant des parties à compléter ; ces parties à compléter sont déterminées par les points où l'ont souhaite mettre en évidence une acquisition. Un commentaire peut être demandé à l'élève pour vérifier qu'il a compris la structure proposée sous la forme du programme à compléter.

La forme "*fermée*" est particulièrement adaptée si par exemple, l'on veut mettre en évidence des difficultés telles que celles qui concernent les calculs sur les ordinaux ; il suffit de laisser à la charge de l'élève une partie de programme comportant un calcul de ce type ; l'éventail des réponses possibles est limité, et l'interprétation de ces réponses peut être faite facilement, en rapport avec les analyses faites antérieurement à l'occasion de EPR1 et EPR2.

Par contre, si l'on veut observer la façon dont l'élève met en place les éléments du programme (variables, structures algorithmiques..) pour utiliser la codification des données résultant de l'énoncé, une forme "*ouverte*" est nécessaire. La difficulté se traduit alors par une écriture erronée. Mais l'interprétation de l'écriture erronée est problématique ; l'écriture peut résulter d'une simple erreur de transcription qui aurait été corrigée dés l'édition sur machine ; elle peut aussi résulter d'une méconnaissance de certaines contraintes de forme liées au langage utilisé, et être corrigée après les premiers essais de compilation ou d'exécution. Nous sommes d'avis de conclure que l'écriture erronée résulte d'une difficulté d'acquisition seulement dans le cas où l'erreur n'est pas détectée après des essais en machine, et persiste lors de la résolution d'exercices analogues. D'autre part, la signification pour l'élève de l'écriture erronée, et le lien avec ses conceptions du dispositif informatique peut ne pas être évidente pour l'observateur, et n'apparaître qu'après plusieurs observations. C'est pourquoi nous avons fait suivre la passation des épreuves EPR1 et EPR2 (chapitres 6 et 7) par des entretiens centrés sur la mise en oeuvre par l'élève des solutions qu'il a produites sur papier ; ce travail nous a permis d'interpréter les erreurs d'écritures relevées lors de l'épreuve sur papier, et de les relier aux conceptions du dispositif.

Un travail approfondi de la même nature que EPR1 et EPR2, concernant un nombre significatif d'élèves serait trop vaste pour entrer dans le cadre de cette thèse ; nous avons dû nous limiter à une épreuve d'une durée d'une heure passée sur papier dans trois classes (une cinquantaine d'élèves). Dans ce cadre, il n'est guère possible de s'éloigner de la forme "fermée", et de multiplier le nombre d'exercices. Nous avons donc choisi de nous limiter aux observations suivantes :

- la compréhension générale de la structure de chaîne ; rôle des arguments dans la fonction sous-chaîne, concaténation...(*D.Sem.Cha.*)
- les difficultés relatives aux cardinaux et aux ordinaux (*D.Ord.Card.*),
- la construction d'une itération portant sur une chaîne de caractères, cette construction étant demandée sous forme "semi-ouverte".

Le premier point est en effet important pour notre étude : l'épreuve doit nous permettre de confirmer que, de façon générale, une structure de données comme les chaînes de caractères pose des problèmes d'acquisition pour des élèves de seconde. Le second point doit nous permettre de confirmer que, la structure ayant fait l'objet d'un début d'acquisition par l'élève, des difficultés particulières peuvent subsister, imposant à l'enseignant de construire des exercices spécifiques. Le troisième point est motivé par la place centrale qu'occupe l'itération dans les objectifs du programme de seconde ; nous voulons voir dans quelle mesure les élèves sont capables de construire

une itération en réponse à un problème simple sur les chaînes de caractères. La question de l'utilisation des types de données (*D.Type*), et la mise en oeuvre des éléments généraux de la programmation (*D.Langage*)n'est donc pas abordée, pour les raisons énoncées ci-dessus.

3. Les conditions de passation et le texte de l'épreuve

L'épreuve a été passée dans deux classes du lycée Victor Hugo de CAEN (classes **D** et **B**), et dans une classe du Lycée Allende d'HEROUVILLE (classe **F**). La passation a eu lieu en fin d'année scolaire 87-88 ; dans les classes **B** et **F**, l'effectif normal de la classe était présent à l'épreuve ; dans la classe **D**, l'épreuve a été passé à la dernière séance de l'année, et seule une partie de l'effectif était présent (on peut penser que les élèves les plus en difficulté ne se sont pas déplacés). Les classes **D** et **B** utilisaient PASCAL (marque BORLAND) comme langage de codage ; la classe **F** utilisait une variété de BASIC structurée et sans numéro de ligne (marque BORLAND également), en fait très proche de PASCAL pour ce qui nous intéresse. La seule différence ayant une influence sur l'énoncé de l'épreuve concerne la déclaration des variables, et l'indication de leur type :

- en PASCAL , cette déclaration doit être faite pour toutes les variables en tête de programme ou dans un bloc (dans ce deuxième cas, la déclaration est valable seulement dans le bloc) ; elle comporte l'indication du type de la variable : Cf [JENSEN WIRTH 74]). Exemple ;

```
var i:integer;
    c:char;
    s: string[255]
```

i est déclaré entier, c caractère, et s chaîne de caractères (255 caractères au maximum).

- en BASIC , il n'y a pas de déclaration de variable ; l'interpréteur crée une variable chaque fois qu'il rencontre dans une expression un identificateur non défini ; le type de la variable est repéré par le dernier caractère de l'identificateur (post-fixe) ; le post-fixe % repère une variable entière, le post-fixe $ repère une variable chaîne. S'il n'y a pas de post-fixe, l'interpréteur donne à la variable le type "réel simple précision". En pratique, on rencontre souvent dans les premiers apprentissages utilisant BASIC des identificateurs sans post-fixe ; en effet, les premiers programmes se font sur des problèmes numériques ; les élèves utilisent ainsi des variables "réelles simple précision", souvent d'ailleurs pour représenter des entiers. Par la suite, la nécessité de post-fixer l'identificateur apparaît à l'occasion des chaînes de caractères.

L'utilisation de BASIC par une classe, et de PASCAL par les deux autres nous a conduit à élaborer deux textes différents ; néanmoins, mise à part la déclaration des variables, ces deux textes diffèrent très peu. Nous donnons ci-dessous la version PASCAL ; on trouvera en annexe à ce chapitre la version BASIC .

3.1 Le problème CLASSE (Exercice 1)

Dans une classe de 12 élèves, un devoir a été noté de 0 à 20 ; le professeur décide de faire un programme lui permettant, en entrant le nom d'un des élèves, d'obtenir en sortie sa note au devoir.

Son programme utilise une chaîne de caractères rangée dans la variable CLASSE et comportant successivement le nom de chaque élève, suivi de la note qu'il (ou elle) a obtenue au devoir (indiquée avec 2 chiffres)

```
PROGRAM LIRE_UNE_CHAINE ;
VAR CLASSE , NOM , NOTE ; STRING[255] ;
     LN , I ; INTEGER;
BEGIN
   CLASSE:='DUPOND09ALFRED12ARTHUR14LAMBERT11MARIE05VICTOR12D
UPONT16DUVAL14CHARLIE19ARMAND13DUBOIS05DUMONT14' ;
   WRITE('Donnez le nom de l''élève  ');
   READLN(NOM);        { entrée du nom d'un élève de la
classe}
   LN:=LENGTH(NOM); { LN est la longueur du nom de l'élève}
   I:=0;
 {début de la boucle de recherche du nom}
-->REPEAT I:=I+1 UNTIL NOM=COPY(CLASSE,I,...);
                  {sortie de la boucle }
-->NOTE:=COPY(CLASSE,...,...) ;
  WRITELN ('La note est ',NOTE )
END.
```

Tu dois compléter les trois emplacements libres (marqués par les '...') de façon à ce que le programme donne bien la note de l'élève.

Ecris dans ta réponse les deux lignes marquées par des flèches, complétées, et explique.

Analyse de l'exercice

Le texte reprend la situation et la structure du premier problème de EPR2 (chapitre 7), avec les mêmes objectifs : observer la façon dont les élèves traitent le second et le troisième argument de la fonction sous-chaîne. Le texte diffère sur un point, la forme choisie pour l'itération : nous avons choisi ici la forme REPETER...JUSQU'A... qui est la plus indiquée (puisque le nombre de passages dans la boucle dépend du nom cherché). Dans EPR2 (chapitre 7), nous avions employé une boucle sur toute la chaîne, et une affectation conditionnelle à l'intérieur de la boucle. Comme nous l'avons expliqué au chapitre 7, la forme REPETER...JUSQU'A...était mal-aisée à construire dans la version de BASIC utilisée par ces élèves ; nous avions ajouté que la forme REPETER...JUSQU'A...pouvait mettre davantage en évidence le paramètre de boucle I, et faciliter la tâche de l'élève.

La première réponse demandée est un cardinal ; la longueur du nom d'élève recherché . La réponse est LN ; on peut penser qu'elle sera trouvée facilement ; elle est compatible avec des conceptions erronées de la fonction LONGUEUR, lui donnant un statut intermédiaire entre l'objet (la chaîne) et le nombre (avec ses aspects cardinaux et ordinaux).

La seconde réponse demandée est la position du premier caractère de la note d'un élève, sachant que la position du premier caractère du nom de l'élève est I ; la réponse correcte est I+LN ; cette position résulte en effet de la translation de LN caractères vers la droite à partir de la position I. Cette réponse suppose une bonne coordination entre l'ordinal I et le cardinal LN, pour maîtriser l'écriture de la translation. Nous avons constaté lors de EPR2 que pour des élèves, le statut flou de LN (en tant qu'ordinal) conduit à lui donner ici une signification intermédiaire entre cet ordinal et le caractère dont il est la position ; il pointerait le dernier caractère du nom de l'élève, indépendamment du fait que le nom de l'élève est une chaîne à part entière ou une sous-chaîne de la chaîne CLASSE. Pour ces élèves, la réponse logique est LN+1 (soit le rang du caractère suivant le dernier caractère du nom de l'élève) ; la réponse LN se rencontre également, marquant un certain flou quant aux rapports entre les aspects ordinaux et cardinaux de la fonction LONGUEUR ; LN c'est "toute la chaîne" donc le dernier caractère serait de rang LN-1. Nous avons également obtenu la réponse I pour la position du premier caractère de la note ; nous avions interprété cette réponse comme l'indication que, pour les élèves qui la produisent, I n'est pas compris comme un index sur les caractères de la chaîne, mais comme un index sur les items (noms, notes) en jeu dans le problème ; nous avions donc relié cette réponse aux difficultés concernant la logique interne de la structure de chaîne.

La dernière réponse demandée est la longueur de la note ; la réponse correcte

est 2 ; elle résulte directement de l'énoncé. Nous avions noté que les réponse erronées sont souvent en rapport avec la réponse I à l'item précédent et concernent également l'acquisition de la logique interne de la structure de chaîne.

3.2 Le problème BONJROU (Exercice 2 question a)

On veut un programme tel que

L'utilisateur entre une chaîne de caractères X, puis un nombre N inférieur à la longueur de X. L'ordinateur affiche une chaîne R formée à partir de X, mais où la dernière lettre de X apparaît avancée en Nième position:

exemple : X: bonjour

N: 5 R ; bonjrou

N: 1 R ; rbonjou

L'idée est de décomposer X en trois parties ;

exemple pour X:bonjour N:4 A: bon B: jou C: r

Complète le programme suivant (lignes marquées par des flèches)

```
PROGRAM TRANSFORMER_UNE_CHAINE;
VAR X,R,A,B,C: STRING[255]; N,L ; INTEGER;
BEGIN
       READLN(X);     {entrée d'une chaîne par l'utilisateur}
       L:=LENGTH(X); { L est la longueur de X }
       READLN(N);     { entrée d'un nombre par l'utilisateur}
-->    A:=COPY(X,...,...);{ A est la première partie de X.}
-->    B:=COPY(X,...,...);{ B est la seconde partie de X }
-->    C:=COPY(X,...,..);{ C est le dernier caractère de X }
-->    R:= .......; WRITELN(R)
END.
```

Analyse de l'exercice

Les trois premiers couples de réponses demandés concernent comme dans la première question les arguments ordinaux et cardinaux de la fonction sous-chaîne ; mais ici, il n'y a pas d'itération ; les difficultés attendues ne viennent pas de l'intervention d'un compteur de boucle ;

- la première ligne se complète en `A:=COPY(X,1,N-1)`. On devrait rencontrer peu de difficulté pour la détermination de la position ; la détermination de la longueur peut poser un problème de coordination entre l'ordinal `N`, et le cardinal demandé.
- la seconde ligne se Complète en `B:=COPY(X,N,L-N)`. De même, la position pose peu de difficulté. Pour la longueur, il faut ici coordonner deux ordinaux ; il s'agit de compter le nombre de caractères entre le rang `N` (*COMPRIS*) et le rang `L-1` (*COMPRIS*) ; les caractères extrêmes étant compris dans le nombre recherché, ce nombre est la différence des ordinaux **PLUS 1**. Il nous semble que ce **PLUS 1** peut être oublié par des élèves pour lesquels la spécificité des ordinaux par rapports aux cardinaux est mal dégagée ; dans ce cas, la réponse serait `L-N-1`.
- La troisième ligne est à compléter pour obtenir le dernier caractère ; la longueur doit donc poser peu de difficultés ; concernant la position, nous avons indiqué ci-dessus que pour certains élèves cette position est `L-1` (au lieu de `L`), la position `L` marquant la fin de la chaîne, donc "pointant" après le dernier caractère.
- La dernière ligne consiste à former le résultat par concaténation des variables `A`,`C` et `B` dans cet ordre. Le problème est peu éloigné du problème **ONJOURB** de EPR1 (Chapitre 7) ; mais ici les élèves n'ont pas à construire eux-mêmes la séparation du mot initial en plusieurs parties dans le but de les concaténer dans un ordre différent, ce qui constituait la difficulté principale de **ONJOURB**. Dans **ONJOURB**, nous avions observé peu de difficultés concernant la concaténation, car les élèves utilisaient en fait très peu cette opération ; nous avons voulu ici imposer l'usage de la concaténation, en imposant l'affectation du résultat à une variable avant son affichage. Ceci doit nous permettre d'observer la façon dont les élèves utilisent la concaténation dans un problème sans difficulté apparente.

3.3 Le problème RUOJNOB (Exercice 2 question B)

On veut cette fois, en utilisant le programme ci-dessus, à partir de la chaîne entrée par l'utilisateur, obtenir la chaîne retournée.

exemple : X : bonjour R : ruojnob

Explique comment tu peux utiliser le programme ci-dessus pour obtenir ce résultat : écris le programme complet pour résoudre B.

Analyse de l'exercice

Notre intention était de demander aux élèves la construction d'un itération simple dans un contexte d'utilisation d'une chaîne de caractères. Dans ce but, il aurait été possible de choisir un problème du type "recherche-translation" comme par exemple le problème **CRYPTAGE** de EPR2 (Chapitre 7). Nous n'avons pas fait ce choix pour deux raisons : d'une part nous avions la possibilité seulement de faire passer une épreuve d'une heure ; or, cette durée est trop courte pour permettre aux élèves d'aborder trois situations différentes. Nous avons donc décidé de réutiliser la situation de la question précédente, ce qui permet aux élèves de gagner du temps sur la compréhension de l'énoncé ; l'autre raison est que certains élèves auraient pu avoir utilisé antérieurement la fonction POSITION (qui rend la position du premier caractère d'une chaîne comme sous-chaîne d'une chaîne donnée) ; cette fonction permet de résoudre les problèmes de recherche translation sans itération.

Nous avons donc fait le choix d'un problème réutilisant la situation de l'exercice précédent ; il suffit pour résoudre, de construire une itération où la donnée N (position dans la chaîne construite du dernier caractère de la chaîne donnée) parcourt toutes les positions de 1 à L-1, le résultat étant affecté à chaque tour de boucle à la variable sur laquelle porte le calcul :

```
L:=LENGTH(X);          { L est la longueur de X }
FOR N:=1 TO L-1 DO
   BEGIN
     A:=COPY(X,1,N-1);{ A est la première partie de X.}
     B:=COPY(X,N,L-N);{ B est la seconde partie de X }
     C:=COPY(X,L,1);{ C est le dernier caractère de X }
      R:= A+C+B;
      X:=R
END
```

Il est possible de voir dans les réponses comment les élèves placent les avertisseurs d'itération ; d'autre-part, certains élèves, ayant positionné ces avertisseurs peuvent ne pas voir la nécessité de l'affectation X:=R (qui permet que la suite du calcul soit faite sur le résultat précédent, et non sur la chaîne initiale) ; ce sera l'indice que la répétition de l'action est comprise, mais que l'évolution des variables (la succession des situations) n'est pas prise en compte.

Nous avons indiqué ci-dessus l'intérêt que présente cet exercice et qui justifie sa présence dans l'épreuve ; en contre-partie, le problème qui y est posé (retournement d'une chaîne) a selon toute probabilité déjà été rencontré par les élèves ; il faut donc que les élèves acceptent de chercher une autre solution à un problème qu'ils ont déjà résolu.

4. Dépouillement des réponses

On trouvera dans le volume Annexes (annexe 7) la liste complète des réponses des élèves aux problèmes. Nous donnons ci-dessous des tableaux récapitulatifs en rapport avec les éléments que nous avons voulu observer, et une analyse de ces résultats. Nous examinons successivement les résultats aux trois problèmes, puis nous mettons en relation la réussite des élèves avec d'autres éléments tels que la décision d'orientation les concernant, et la classe où ils ont travaillé.

4.1 Les réponses au problème CLASSE[101]

4.1.1 L'item "longueur du nom à rechercher"

On trouve 5 réponses erronées:

1 (2 fois) , 7, NOTE, LEN(Note)-1 , LEN(NOM)-1

4.1.2 Le couple d'items : "position de la note, longueur de la note"

	position correcte (I+LN)	position LN	position LN-1 ou LN-2	position I	position NOM	position NOM$+LN	totaux
Longueur correcte	18	5	2	1		1	27
longueur I		6			3		9
longueur LN				2	3		5
longueur NOM				4			4
autres				5			5
totaux	18	11	2	12	6	1	50

4.1.3 Analyse et comparaison avec EPR2.

La réussite des élèves

Dans les deux épreuves, les élèves qui réussissent sont en minorité ; 2/13 pour EPR2, 18/52 ici. Dans les deux épreuves également, les erreurs concernent peu la longueur du nom ; en effet, la réponse correcte est compatible avec des interprétation erronées, telle que : LN veut dire "prendre le nom tout entier..." Les erreurs concernant la longueur de la note sont plus fréquentes que dans EPR2 (2/13 dans EPR2, 24/52 ici) On peut faire l'hypothèse que pour beaucoup, parmi ces 24 élèves, le rôle des différents arguments de la fonction sous-chaîne n'est tout simplement pas compris. Concernant la position de la note dans la chaîne, on retrouve les mêmes erreurs que dans EPR2, mais avec des fréquences d'apparition différentes :

Les réponses LN+1 ou LN ou LN-1 ou LN-2 pour la position

Elles apparaissent ici pour 13 réponses sur 52, et dans EPR2, pour 8 réponses sur 13 . Nous avions donné pour cette erreur, l'interprétation suivante : LN apparaît aux élèves comme une référence ordinale marquant le dernier caractère du nom (pour les réponses LN+1), ou son suivant (pour les réponses LN) indépendamment des caractères qui précédent ce nom dans la chaîne. Cette interprétation nous avait paru cohérente avec la conception de la fonction longueur comme représentant "La chaîne toute entière", cette conception étant à relier au double statut ordinal et cardinal du résultat de cette fonction.

Nous avions noté dans EPR2 que la réponse LN pour la position du premier caractère de la note s'observe particulièrement chez les élèves qui donnent comme rang du dernier caractère d'une chaîne, la longueur de cette chaîne moins un, comme si, en tant qu'ordinal, cette longueur pointait un caractère au-delà de la chaîne. Ici, on observe seulement des réponses de ce type. Il se trouve que le problème suivant (**BONJROU**) demande de calculer le dernier caractère d'une chaîne ; l'examen des réponses des élèves montre que ceux qui donnent une réponse LN, ou approchante pour la position de la note, donnent à une exception prés, la position LN-1 pour le dernier caractère d'une chaîne. L'observation obtenue à la suite de EPR2 se trouve donc ici confirmée.

Toujours dans le cas d'une réponse LN pour la position de la note, on observe ici la réponse I dans un cas sur deux, alors que l'on obtenait systématiquement la réponse correcte (2) dans EPR2. Un élève explique ainsi sa réponse ; «*il faut que je demande la longueur du nom et l'identité de l'élève afin de lui donner sa note*». Il n'y a donc pas ici

---

[101] un élève a répondu par la mention "*impossible*" ; un autre a traité l'exercice de façon entièrement numérique ; ils ne seront pas comptabilisés dans les deux tableaux suivants.

de compréhension de l'itération qui précède, ni du rôle des différents arguments ; chacune des valeurs données à ces argument est comprise comme une indication au statut imprécis, mais l'ensemble est considéré par l'élève comme suffisant pour que l'ordinateur produise la note.

Au contraire, les réponses où la longueur est 2 traduisent une compréhension du rôle des arguments et de la structuration des données, au delà de l'erreur de conception des ordinaux qui entraîne le réponse `LN` pour la position de la note ; un élève l'exprime ainsi : «*On veut la note, donc on prend les caractères mis après le nom d'où* `LN` *et on prend 2 caractères après pour avoir la note*». On peut aussi penser que l'élève a compris `I` comme un pointeur qui, en se déplaçant dans la chaîne, marque l'origine pour l'ordinal, second argument dans la fonction `sous-chaîne`. Nous considérons que, contribuant à la même anticipation, les deux compréhensions peuvent coexister chez un élève : elles se rapportent toutes les deux à un schéma mental où les ordinaux sont considérées comme des objets intrinsèquement liés aux chaînes sur lesquelles ils servent de pointeurs, et non comme de simples quantités numériques (un pointeur sur une chaîne s'illustre par l'image familière d'un doigt fixant un caractère sur une feuille imprimée ou du curseur sur l'écran texte d'un ordinateur ; dans cette métaphore, le pointeur est bien plus qu'une quantité numérique. On peut penser que la conception qui conduit à la réponse `LN` ou à une réponse approchante, est proche de ce schéma de pensée).

La réponse `I` pour la position

On observait deux réponses de ce type dans EPR2, associées à la réponse `LN` pour la longueur de la note. Nous avions montré que pour ces deux élèves, `I` ne parcourt pas l'ensemble des positions des caractères dans la variable chaîne CLASSE, mais constitue une sorte d'index repérant les élèves ; ainsi cette réponse indique l'intention de l'élève d'obtenir la note de l'élève `I`, sans compréhension de ce que `I` représente en tant que position. Ici, 13 réponses sur 52 donnent `I` pour la position de la note ; elles sont associées à des réponses assez diverses pour la longueur ; comme pour le couple de réponse (`LN` , `I`) ce type de réponse montre une incompréhension du rôle respectif des arguments de la fonction sous-chaîne : l'élève s'assure simplement de ce que l'ordinateur dispose de suffisamment d'informations pour trouver la note.

Les réponses `NOM` pour la position

elles s'apparentent aux réponses `LN`, avec en plus une confusion de type (la chaîne est assimilée à sa longueur) ; elles n'était pas présentes dans EPR2. On notera qu'elles s'accompagnent de réponses erronées pour la longueur de la note, et dénotent donc également une incompréhension du rôle respectif des arguments de la fonction sous-chaîne, et donc de la structuration des données dans le problème.

Récapitulation

Les pourcentages sont donnés à titre indicatif pour faciliter la comparaison.	EPR2		EPR3	
Réponses correctes	2	15%	18	35%
Erreur seulement sur la position de la note	9	70%	10	20%
Non compréhension de la structure de données	2	15%	22	42%
divers			2	4%
total	13	100%	52	100%

On constate que pour EPR2, la catégorie majoritaire était constituée d'élèves ayant compris l'utilisation de la structure pour le problème posé, mais qui rencontraient une difficulté spécifique ayant trait à l'utilisation des ordinaux dans le parcours itératif d'une chaîne. Ici, cette catégorie existe, mais en pourcentage plus faible ; ceux qui réussissent sont plus nombreux proportionnellement, mais il existe aussi une fraction importante des élèves à qui l'utilisation de la structure pour le problème posé échappe totalement.

4.2 Les réponses au problème BONJROU

4.2.1 Les erreurs dans le calcul de chacune des 3 parties (A, B, C):

Le tableau suivant donne, pour chacune des trois parties le nombre de réponses erronées sur la position, le nombre de réponses erronées sur la longueur, puis les couples de formulations erronées les plus fréquents.

	A (ou A$) du premier au N-1ème caractère		B (ou B$) du Nème à l'avant dernier caractère		C (ou C$) le dernier caractère	
	effectifs	pourcentages	effectifs	pourcentages	effectifs	pourcentages
position erronée	23	44%	34	65%	36	69%
longueur erronée	34	65%	45	87%	14	27%
au moins une erreur (ou pas de réponse)	35	67%	45	87%	40	77%
couples de formulations erronées les plus courants	couples / effectifs		couples / effectif		couples / effectifs	
	L , N	8	L+N , N	4	L-1,1	17
	1 , N	3	N ,L-N-1	4	N , 1	3
	1 , L/2	4	N , L-1	4	L , N	3

Le calcul de la première partie (A ou A$) pouvait apparaître comme relativement simple ; démarrant au premier caractère du mot, la position est 1, et la longueur est N-1, de façon que le caractère suivant immédiatement cette partie soit le Nième. Les erreurs sont pourtant nombreuses. Elles peuvent résulter de formulations ne prenant pas en compte les contraintes ; on retrouve dans ce cas des couples 2ème, 3ème argument que l'élève écrit pour "donner à l'ordinateur toutes les informations nécessaires" au calcul de la sous-chaîne ; par exemple le couple (L,N) pourrait signifier : "dans le mot tout entier, prendre jusqu'au Nième caractère.

D'autres réponses se caractérisent par un calcul correct de la position (1) mais une position erronée, le plus souvent N au lieu de N-1 ; on peut faire l'hypothèse que dans ce cas, le sens des arguments de la fonction sous-chaîne est compris, mais que l'élève éprouve des difficultés à coordonner cardinaux et ordinaux pour trouver l'argument longueur. Un seul élève donne la position 0 pour le premier caractère.

Le calcul de la seconde partie (B ou B$) : comme pour A, le calcul de la position est relativement simple ; par contre, la longueur de cette partie est une différence d'ordinaux (L-N), dont nous avons indiqué ci-dessus les pièges. Les erreurs sont particulièrement nombreuses ; parmi celles-ci, des couples tels que (N, L-N-1) témoignent seulement d'une erreur de calcul de la longueur.

Le calcul du dernier caractère (C ou C$) est classique, et a pu être déjà rencontré par les élèves. La longueur ne devrait pas poser de problèmes ; elle est erronée pour 27% des élèves, ce qui doit correspondre à des élèves n'ayant pas intégré le rôle respectif de chaque argument de la fonction sous-chaîne.

Parmi les 26 élèves donnant la longueur correcte (1) mais une position erronée, 17 donnent la réponse L-1 pour cette position ; seul un de ces élèves donne la position 0 pour le début de la chaîne, ce qui serait cohérent avec la position L-1 pour le dernier caractère. Nous avons rencontré antérieurement le comportement consistant à donner la position L-1 pour le dernier caractère tout en donnant la position 1 pour le premier ; nous l'avons attribué à une difficulté pour coordonner la signification cardinale de la fonction longueur, avec la numérotation ordinale des caractères de la chaîne : signifiant "toute la chaîne", L "pointerait" en tant qu'ordinal "au delà" du dernier caractère. Ceci est renforcé par une compréhension de l'écriture L-1 comme traduction de **l'action** "*séparer le dernier caractère*" (*D.Langage*). Nous avons souligné lors du dépouillement du problème **CLASSE**, que ces élèves attribuent la position LN (au lieu de LN+1) à la position du caractère qui suit un mot de longueur LN

dans une chaîne.

Quelques réponses comportent une erreur de type qui se caractérise par la présence d'un identificateur de variable chaîne dans le calcul d'un ordinal ou d'un cardinal ; ce type de réponse est représentatif de la confusion entre un caractère et sa position dans une chaîne : `(A+1, L-(A+1)-1)`, par exemple, où l'élève désigne par `A + 1` la position du caractère qui suit la chaîne `A`. Ainsi 7 réponses comprennent l'identificateur de variable chaîne `A` (ou `A$`), 5 réponses comprennent `B` (ou `B$`), 1 réponse comprend `C`, et 2 réponses comprennent `R`.

Répartition des réponses selon le type d'erreur dans le calcul de `A, B, C`

	`A` (ou `A$`) du premier au `N-1`ème caractère		`B` (ou `B$`) du `N`ème à l'avant dernier caractère		`C` (ou `C$`) le dernier caractère	
	effectifs	**pourcentages**	**effectifs**	**pourcentages**	**effectifs**	**pourcentages**
réponses correctes	17	33%	7	14%	12	23%
erreur, mais rôle des arguments compris	10	19%	9	17%	17	33%
rôle des arguments non compris	22	42%	33	63%	22	42
pas de réponse	3	6%	3	87%	1	2%

On remarque que le nombre de réponses s'interprétant comme une non-compréhension du rôle des arguments augmente de façon très significative pour le calcul de `B` ; la compréhension qui paraît attestée par les réponses concernant `A` et `C`, est remise en cause par l'augmentation de la difficulté que représente le calcul de `B` ; l'observation confirme donc que `A` et `C` constituent des cas particuliers pour lesquels une réponse peut se révéler compatible avec certaines conceptions erronées.

4.2.2 Le calcul du résultat (variable `R` ou `R$`) :

formulations correctes `(A+C+B...)`	18	35%
restitution de la chaîne initiale : `A,B,C` ou `copy(A,B,C)` ou `A+B+C` ou `A$+B$+C$`	12	23%
formulations utilisant "`INSERT`":	4	8%
`C$+A$+B$` ou `C,A,B`	3	6%
formulations ne comprenant pas l'une des variables `A, B, C` .(ou `A$,B$, C$`)	7	14%
pas de réponse	5	10%
autres erreurs	2	4%

72 % des réponses associent `A,B` et `C` ; on peut penser que pour ces réponses, le problème a été compris.

8% des réponses utilisent l'instruction `INSERT`[102]. Cette instruction est très mal utilisée: on trouve en effet diverses erreurs de syntaxe en particulier l'utilisation

---

[102] L'instruction `INSERT` est présente dans la variété de PASCAL (TurboPASCAL de marque BORLAND) utilisée par deux des classes ayant passé l'épreuve : `INSERT(source,s,index)` *insère la chaîne* `source` *dans la chaîne* `s` *à partir de la position* `index`.(Manuel de Turbo PASCAL Version IV) . Cette instruction modifie un de ses arguments. Il s'agit donc d'une **action**, au sens que nous avons donné au chapitre 5. Nous avons fait remarquer, que, en général, dans les langages impératifs évolués, les actions simples sont écrites à l'aide de l'affectation et de la notation fonctionnelle ; ainsi, l'action de faire la somme de `a` et `b` et d'affecter le résultat à `a` s'écrit `a:=a + b`, plutôt que `sommer(a,b)`. L'instruction `INSERT` ne suit pas cette usage: en fait, `INSERT(source,s,index)` est équivalent à

```
source:=sous-chaîne(source,1,index-1)+s
        +sous-chaîne(source,index,longueur(source)+1)
```

de cette instruction comme une fonction (alors qu'elle ne rend pas de résultat, mais modifie son second argument). Par exemple, la formulation `R=A+B+INSERT(C,R,N)` employée par deux élèves traduit directement l'idée : «*concaténez* `A` *et* `B`, *puis insérez* `C` *dans le résultat à la position* `N`». Il s'agit donc une formulation marquée par la conception naïve du langage de programmation.

20% des réponses donnent `A, B, C` dans cet ordre, les variables pouvant être reliées par le signe de concaténation (+ , en BASIC comme en PASCAL) ou dans d'autre cas par un lien plus vague, matérialisé par des virgules. On peut s'étonner de cette réponse, car `A+B+C` a pour valeur la chaîne initiale ; un élève précise : «`A,B,C` *mais dans le désordre*» . Nous pensons qu'ici les élèves déléguent à l'ordinateur le soin de l'ordre dans lequel associer `A, B,` et `C` : il est en effet possible qu'ils imaginent l'ordinateur capable de deviner l'intention du programmeur à partir du calcul de `A, B` et `C`, leur idée implicite étant que l'ordinateur n'est certainement pas assez bête pour redonner la chaîne initiale.

4.2.3 Comparaison des performances dans le problème CLASSE et dans le problème BONJROU

Comme pour le problème **CLASSE**, nous répartissons les élèves en trois catégories selon leurs performances au problème **BONJROU**: ceux pour lesquels la structure de donnée n'est pas comprise, ceux pour lesquels la structure est comprise mais dont les réponses comportent des erreurs, soit dans le calcul des arguments, soit dans la concaténation, et ceux dont la réponse est exacte. Le tableau ci-dessous permet d'estimer la cohérence des performances dans le problème **CLASSE** et dans le problème **BONJROU**.

	performances au problème **CLASSE**			
performances au problème **BONJROU**	structure non comprise	structure comprise mais erreur dans les arguments ordinaux	réponse exacte	total
structure non comprise	21	5	5	31
structure comprise mais erreur dans les ordinaux ou la concaténation	4	4	11	19
réponse exacte	0	0	2	2
total	25	9	18	52

Le tableau montre une relative cohérence des résultats dans les deux questions, avec cependant une baisse générale des performances dans le problème **BONJROU**. A travers ses 7 items, cette question recèle une accumulation de difficultés qui contribuent à faire échouer y compris certains de ceux qui réussissaient totalement le problème **CLASSE**.

4.3 Les réponses au problème RUOJNOB

Dans le tableau ci-dessous , nous classons dans un premier groupe les réponses qui traitent effectivement le problème, en ce sens qu'elles utilisent, comme l'énoncé le demande, la décomposition en trois sous-chaînes, et le calcul d'une nouvelle chaîne, où le dernier caractère est passé en Nième position. Dans ce groupe, les élèves ont pu, ou non construire une itération , et cette itération peut être correcte ou erronée. Nous classons dans un deuxième groupe ceux qui tentent, éventuellement avec succès, la construction d'un programme d'inversion d'une chaîne de caractères, mais sans utiliser le calcul de la question précédente.

réponse correcte :	2	4%
problème compris, mais erreur dans l'itération	5	10%
séparation en au moins 3 sous-chaîne, mais pas d'itération	7	13%

total premier groupe (problème effectivement traité)	14	27%

algorithme d'inversion correct mais sans rapport avec l'énoncé	11	21%
algorithme d'affichage des caractères dans l'ordre inverse	2	4%
algorithme sans rapport avec l'énoncé (erroné)	9	17%
retournement par moitié	2	4%
total second groupe (utilisation algorithme sans rapport avec l'énoncé)	24	46%

pas de réponse	14	27%

Une moitié des élèves ne répond pas à la question posée, c'est-à-dire qu'ils utilisent un algorithme classique de retournement avec itération, probablement celui construit en travail dirigé. Les formes que prend cet algorithme varient selon les élèves : deux élèves "retournent" séparément les deux moitiés de la chaîne, avant de les recomposer à l'envers ; ils avaient compris la question précédente comme le calcul des deux moitiés de la chaîne, de façon à insérer le dernier caractère au milieu ; ils ont voulu réutiliser ici la séparation de la chaîne en deux moitiés.

Conclusions sur le problème **RUOJNOB**

Les 14 élèves du premier groupe distingué dans le tableau ci-dessus, réutilisent effectivement la décomposition en trois parties de la chaîne ; parmi eux, une petite moitié ne construit pas d'itération ; quand ils donnent effectivement une valeur au résultat, il est de la forme C+B+A, de sorte que les 3 parties sont rendues "à l'envers" (mais à l'intérieur des parties, l'ordre n'est pas modifié). 7 élèves (soit 15%) insèrent effectivement le calcul qu'ils ont produit à la question précédente, dans une itération où N parcourt les positions dans la chaîne. Seuls deux élèves affectent à chaque tour de boucle le résultat de ce calcul à la chaîne sur laquelle porte le calcul. Les réponses des autres élèves omettent cette partie du corps de boucle ; ainsi, seul le dernier tour de boucle produit une modification du résultat.

Pour les élèves qui répondent effectivement à la question, **une première difficulté est donc d'envisager le problème comme itératif**. Ensuite, dans la construction de l'itération, il faut construire le corps d'itération comme une **instruction modifiant la chaîne sur laquelle elle porte** (prise en compte des situations successives, et pas seulement des actions)[103] ; la presque totalité des élèves échoue à ce niveau.

Les élèves du second groupe traitent ce problème comme une question de cours sur le retournement d'une chaîne : ils représentent une moitié de l'effectif. Ces élèves n'ont sans doute pas la maturité suffisante pour considérer plusieurs solutions à un problème, ce qui semble important à prendre en compte pour la construction d'une épreuve ; la question perd en effet un peu de son intérêt car on ne peut savoir ce qu'auraient répondu ces élèves s'ils n'avaient eu déjà connaissance d'un algorithme de retournement d'une chaîne.

5. Comparaison des performances des élèves selon la décision d'orientation du conseil de classe, et la classe dont ils sont issus

5.1 Intérêt d'une classification des résultats selon la décision d'orientation

La question des rapports entre les capacités d'acquisition en informatique et dans

[103] Une conduite de ce type a été mise en évidence par [SAMURCAY 85] au cours le programmation d'algorithmes de calculs mathématiques (voir chapitre 2)

les autres disciplines scientifiques (et particulièrement les mathématiques) est souvent posée, bien qu'on ne dispose pas d'éléments précis à ce sujet. [BARON 89] a mis en évidence d'une part le taux important de non-continuation de l'option en classe de première (60% en 85/86), et montré que, à partir de la classe de première, les élèves des classes scientifiques (première S, terminales C et D) sont très largement sur-représentés dans l'option. Mais ceci n'a jamais été mis en relation avec les acquisitions des élèves à l'issue de la seconde ; la non-continuation de l'option après la seconde pourrait donc s'expliquer aussi bien par un constat d'échec du point de vue des acquisitions, que par des contraintes administratives ou des politiques d'établissement[104] . Une autre hypothèse a été envisagée et testée par [BARON 89] : l'image de l'informatique, pour les élèves qui abandonnent, se caractérise par "*un surcroît de travail*", "*une discipline peu intéressante*", ou "*trop rigoureuse*". Ces formulations sont compatibles avec des difficultés d'acquisition, mais n'en constituent pas des preuves objectives.

L'épreuve est passée en fin d'année ; à cette période, les décisions d'orientation du conseil de classe sont connues. Elles peuvent consister en une orientation en classe de Première scientifique (Première S), en une autre classe de Première (Première B : sciences économiques ou Première G : gestion) ou un redoublement. Il n'y a pas, dans les trois classes, d'élèves concernés par une orientation en Première littéraire (Première A). Nous prenons l'orientation en Première S comme l'indice d'une classe de seconde réussie dans les disciplines scientifiques. Les autres orientations, qu'ils s'agisse d'un redoublement ou d'une orientation vers une autre classe de Première ne sont pas nécessairement la conséquence d'un échec caractérisé dans les disciplines scientifiques ; nous considérons néanmoins qu'il existe une bonne probabilité pour que ces orientations concernent des élèves réussissant moins bien dans les disciplines scientifiques. C'est pourquoi nous choisissons de mettre en rapport les performances aux différents exercices de l'épreuve, et l'orientation ou non vers une classe de Première scientifique. Pour 2 élèves, l'orientation n'a pas pu être connue.

5.2 Mise en évidence de différences entre classes

Du fait du manque de tradition en enseignement de l'informatique, l'effet des stratégies d'apprentissage, en terme de structures de programmation enseignées, et d'exercices proposés à la résolution, est très peu connu. Il est donc intéressant de saisir l'occasion d'une épreuve passée sur plusieurs classes pour tenter de mettre en évidence des variations entre les trois classes, et de mettre en relation ces variations avec ce que nous pouvons connaître des particularités de l'enseignement dans chaque classe, et des caractéristiques propres aux élèves.

5.3 Le choix d'un critère de réussite à l'épreuve

Cette classification selon la décision d'orientation et le repérage de différences entre classes, suppose que la réussite à l'épreuve fasse l'objet d'une quantification. Nous avons cherché à définir pour chaque élève, et pour chaque exercice un "*indice de réussite*" significatif d'un niveau de maîtrise, et donc repris sous forme chiffrée, la classification distinguant trois niveaux : celui où la structuration des données proposée dans l'énoncé n'est pas comprise, celui où cette structuration est comprise, mais où des difficultés à manipuler cette structure se manifestent, et enfin le niveau où la réponse est correcte:

- 0 est attribué quand l'énoncé n'a pas été compris, quand l'utilisation des fonctions sur les chaînes montre que la structuration des données n'est pas comprise, ou quand il n'y a pas de réponse. En particulier, nous avons attribué 0, pour le problème **RUOJNOB** aux élèves qui produisent un algorithme de retournement non conforme à l'énoncé, même s'il est correct ; en effet, ce type de réponse montre que l'énoncé n'a pas été compris et d'autre part, même s'il est correctement restitué, nous n'avons pas de garantie que l'algorithme soit réellement compris par l'élève.

---

[104] Des établissements pourraient décourager les élèves non-scientifiques de continuer pour réserver des places au élèves de première S, le fait d'avoir suivi l'option pouvant être considéré, pour ces élèves, comme un supplément intéressant dans la perspective d'études post-baccalauréat

- 1 est attribué quand l'énoncé a été compris, quand l'utilisation des chaînes pour la codification est comprise, mais qu'il y a des erreurs dans l'utilisation de la structure de chaîne. Nous avons attribué 1 au problème **CLASSE** si la solution comporte seulement une erreur dans le calcul de l'ordinal. Nous avons attribué 1 au problème **BONJROU** pour des erreurs concernant la concaténation, ou les calculs d'ordinaux ou de cardinaux, le sens des arguments de la fonction sous-chaîne étant cependant compris. Dans le problème **RUOJNOB**, nous avons attribué 1 si l'utilisation dans une itération du programme de la question A est comprise, et s'il y a des erreurs dans la construction de cette itération (par exemple l'oubli de la mise à jour de la variable itérative).
- 2 est attribué quand l'exercice est correctement résolu .

La somme des trois indices aux trois problèmes est un nombre compris entre 0 (échec dans les trois problèmes) et 6 (réussite totale). Nous distinguons par la suite trois *groupes de réussite* : ceux dont la somme est 0 ou 1, ceux dont la somme est 2 ou 3, ceux dont la somme est 4 ou 5 (il n'y a pas d'élève obtenant une somme égale à 6). Il est clair qu'ainsi le premier groupe sera constitué essentiellement d'élèves pour lesquelles la structuration des données à l'aide des chaînes n'est pas comprise ; le second groupe sera constitué par les élèves présentant un début d'acquisition des fonctions sur les chaînes, mais rencontrant des difficultés ponctuelles ; le troisième groupe sera constitué des élèves dominant suffisamment l'emploi des chaînes. Nous utiliserons également, pour caractériser la réussite moyenne dans une population d'élèves, un "indice moyen de réussite", constitué de la somme des indices de chacun des élèves divisé par l'effectif de la population. En annexe au chapitre 8 (Volume Annexes) on trouvera, en regard des réponses de chaque élève, l'indice de réussite que nous lui avons attribué.

5.4 Réussite à l'épreuve et décision d'orientation

Le tableau suivant donne la répartition des élèves suivant la décision d'orientation et suivant le *groupe de réussite*, ainsi que l'indice moyen pour chaque type d'orientation:

groupe de réussite	orientation 1S		autre orientation		total	
4-5	5	16%	1	5%	6	12%
2-3	12	39%	2	11%	14	28%
0-1	14	45%	16	85%	30	60%
total	31	100%	19	100%	50	100%
indice moyen	1,9		0,7		1,4	

Une proportion plus importante d'élèves orientés en Première S réussit au moins partiellement (55% contre 16%) ; néanmoins, l'orientation en Première S ne semble pas constituer une garantie de réussite (45% échouent) ; de même, certains élèves non orientés en Première S peuvent réussir. L'indice moyen de réussite (somme des indices sur nombre d'élèves) confirme cette conclusion.

5.5 Réussite à l'épreuve et classe d'origine des élèves

La réussite pour chaque type d'orientation varie notablement d'une classe à l'autre comme le montre le tableau suivant :

Indice moyen de réussite dans chaque classe pour chaque orientation

	orientation 1ère S		autre orientation		ensemble de la classe	
	effectif	indice moyen	effectif	indice moyen	effectif	indice moyen
Classe B	13	0,7	13	0,3	26	0,5
Classe D	7	2,1	1	2	9	2,1
Classe F	11	3,1	5	1,4	17	2,5

note : dans chacune des classes D et F, un des élèves n'a pas d'orientation connue.
Commentaire du tableau:

L'épreuve a été passée dans trois classes différentes (Classe **B**, classe **D**, classe **F**). La comparaison dans la classe **D** n'offre pas d'intérêt à cause de la faiblesse de l'effectif hors orientation Première S (dans cette classe, l'épreuve a été passée au cours d'une des toutes dernières séances de l'année ; de nombreux élèves étaient absents, en particulier sans doute ceux qui pensaient ne pas réussir à l'épreuve) ; la classe **F** comprend 4 élèves fort brillants (selon le professeur) ; ce sont les seuls élèves qui traitent vraiment le problème **RUOJNOB**; ils contribuent à tirer vers le haut l'indice de réussite moyenne dans le groupe de ceux qui sont orientés en Première S ; mais le groupe des orientations hors Première S de la classe **F** a une meilleure performance moyenne que le groupe orienté en Première S dans la classe **B**.

5.6 Cours et exercices sur les chaînes, pratiqués dans les différentes classes

Nous nous proposons d'examiner si les différences de performances entre classes peuvent s'expliquer par le contenu des cours, et les énoncés des exercices proposés par le professeur de chacune des classes. Nous avons pu disposer d'un cahier d'élève des classes **B** et **D**, et du cahier de préparation du professeur de la classe **F**. Il résulte de l'étude de ces documents que :

- les classes **B** et **D** ont suivi la même progression de cours et d'exercices, les deux professeurs préparant ensemble ces activités ; le cours porte sur "*le type caractère*" puis "*le type chaîne*" ; il occupe un des 5 chapitres du cahier ; les chapitres étant d'égale importance (à l'exception du chapitre sur l'itération, un peu plus volumineux que les autres) on peut estimer à une dizaine de séances de 1h à 1h30 (sur la soixantaine que compte l'année), le temps consacré aux caractères et aux chaînes. Le jeu d'instruction et de fonctions indiqué aux élèves est très étendu ; il comprend :
  - les fonctions de conversion entre caractères et ordinaux (dans le code `ASCII`[105]) ; ainsi que les fonctions "`précédent`" et "`successeur`" dans l'ordre induit par ce code,
  - le jeu complet de fonctions sur les chaînes ("`sous-chaîne`", "`concaténation`", "`longueur`", "`position`"),
  - un jeu d'instructions sur les chaînes modifiant leur argument, propre à Turbo-PASCAL (la variété de PASCAL de Marque Borland utilisée par ces classes) : l'instruction `INSERT` signalée plus haut (paragraphe 4.2.1) et l'instruction `DELETE`[106],
  - un jeu d'instructions de conversion entre un entier et la chaîne de sa représentation en base 10.

  D'autre part, l'équivalence entre la structure de chaîne et la structure de tableau de caractères (propre à PASCAL) a été signalée, et utilisée : une variable déclarée par exemple de type `string[255]`, est un tableau de caractères à 255 éléments ; l'élément d'ordre `0` est le caractère dont le code est la longueur actuelle de la chaîne ; l'élément d'ordre `i` contient le caractère de rang `i` de la chaîne, si `i` est compris au sens large entre `1` et la longueur de la chaîne.

  Les exercices suivants ont été traités dans les classes **B** et **D**:
  - entrer `nom` et `prénom` ; sortir une chaîne identité (`nom` et `prénom` séparés par des espaces).
  - entrer deux chaînes ; les sortir dans l'ordre alphabétique.
  - retourner une chaîne (au sens de l'exercice **RUOJNOB**), (utilisation de la

---

105 Dans un dispositif informatique donné, l'emploi de caractères du jeu alphabétique usuel, souvent étendu à des caractères de commande (saut de ligne, effacement...) suppose le codage de chaque caractère sous la forme d'une entité mémorisable. Dans les micro-ordinateurs, la mémoire peut être considérée comme composée d'éléments prenant 256 valeurs (ce sont des *octets*). C'est pourquoi un jeu étendu de 256 caractères est utilisé, la correspondance étant normalisée sous le terme de **code ASCII**, `ASCII` étant les initiales de `American Standard Code Information Interchange`. Voir [MORVAN 81]

106 `DELETE(mot, pos, nbr)` enlève `nbr` caractères à la variable chaîne `mot`, à partir de la position `pos`

fonction sous-chaîne).
    - entrer une chaîne ; sortir la même chaîne, les occurrences du caractère `'O'` étant supprimées (utilisation de la fonction `POS`, et de l'instruction `DELETE`).
    - entrer une chaîne ; sortir la même chaîne, les occurrences du caractère `'O'` étant remplacées par le caractère `'X'` (utilisation de l'instruction `INSERT`).
    - entrer une chaîne ; vérifier qu'il s'agit de la représentation en base 10 d'un nombre entier ; (utilisation de l'instruction de conversion chaîne-entier).
    - retourner une chaîne (au sens de l'exercice **RUOJNOB**), (utilisation de la structure "tableau de caractères").
- la classe **F** a consacré 11 séances de cours ou de travaux pratiques aux chaînes de caractères, ce qui correspond sensiblement au même volume horaire que les deux autres classes ; ont été présentées :
    - les "opérations " ; égalité et concaténation.
    - les fonctions de conversion entre caractères et ordinaux (dans le code `ASCII`), la fonction d'entrée d'un caractère au clavier,
    - les fonctions de conversion entre un entier et la chaîne de sa représentation en base 10.
    - les fonctions sous-chaîne et longueur, mais aussi des fonctions spécifiques à BASIC (`LEFT$` et `RIGTH$`) qui rendent des sous-chaînes particulières (respectivement sous-chaîne à partir de la gauche, et sous-chaîne à partir de la droite).

  Les exercices traités par la classe **F** ont été les suivants ;
    - codage `ASCII` : "*à une lettre associer son code*", "*à un code associer un symbole*"
    - *écrire un algorithme qui compte les* "e" *d'un mot*
    - *écrire un algorithme qui permet d'enlever tous les* "e" *d'un mot*
    - *reconnaître si un mot est un palindrome*
    - traduire un mot en "*javanais*" (tous les "a" sont remplacés par "ava", les "e" par "eve" etc... pour toutes les voyelles)
    - écrire un programme de "*codage*" d'une chaîne ("*décalage identique pour toutes les lettres, ou établir une table de correspondance*"), écrire le programme de décodage.

5.7 Relation entre la réussite des élèves et les activités en classe

Le jeu de fonctions et d'instructions paraît très volumineux dans les trois classes, particulièrement dans les classes **B** et **D**, surtout si on le rapporte au petit nombre de séances consacrées à la structure de chaîne. Il est très largement redondant ; `DELETE` et `INSERT` de Turbo-PASCAL par exemple, se réalisent avec la fonction `sous-chaîne`, la fonction `longueur` et la `concaténation`, et cette réalisation, en réponse à des problèmes, est en elle-même une activité intéressante. Il en est de même pour les fonctions `LEFT$` et `RIGHT$` de BASIC . Un jeu d'instructions un peu moins abondant dans la classe **F** est à mettre en relation avec une meilleure réussite à l'épreuve dans cette classe, sans toutefois qu'on puisse dire s'il s'git d'un élément déterminant.

On peut penser que les professeurs ont préparé ce cours avec l'hypothèse que la structure serait facilement acquise par les élèves, et ont accumulé les fonctions et instructions pour donner du contenu à cette partie de leur cours. Les résultats globaux à l'épreuve montrent que cette hypothèse n'est pas vérifiée. L'épreuve supposait seulement la connaissance du jeu réduit d'instructions (`sous-chaîne`, `longueur`, `concaténation`) ; certains élèves ont tenté l'utilisation de l'instruction `INSERT` , mais leurs réponses montrent que cette instruction n'est pas comprise (voir paragraphe 4.2.2). La correspondance entre caractères et nombres dans le code `ASCII` sur laquelle on insiste dans les trois classes nous paraît liée à une réalisation de la structure sur un type de machine plutôt qu'à une propriété des chaînes. Elle risque d'interférer avec la correspondance entre nombre et la chaîne de sa représentation en base 10, qui nous semble plus fondamentale.

Les exercices résolus dans les classes avec la fonction `sous-chaîne` utilisent en fait seulement l'accès à un caractère de la chaîne (le troisième argument est donc systématiquement 1). Les exercices utilisant l'itération sont en fait tous du même type: on parcourt une chaîne caractère par caractère, et on recompose une chaîne résultat au fur et à mesure en utilisant l'information lue dans la chaîne donnée. La compréhension de la structure ordinale, et surtout des rapports entre cardinaux et ordinaux est donc peu sollicitée. De même, la concaténation est utilisée dans un contexte où elle peut être noyée dans les difficultés propres à l'itération.

6. Conclusions

Les réponses à l'épreuve confirment les difficultés d'acquisition de la structure de chaîne, attestée en premier lieu par le nombre important de réponses marquées par une conception naïve du langage: 50% dans le problème **CLASSE** et 60% dans le problème **BONJROU**. Dans le meilleur des cas, ces réponses sont constituées de formulations où "*toutes les informations sont données à l'ordinateur*", mais qui ne prennent pas en compte la nature du résultat ou celle des arguments des fonctions.

Le contenu des cours et exercices des différentes classes où cette épreuve a été passée permet de penser que l'acquisition de la structure a été jugée par les enseignants comme ne devant pas poser a-priori de difficulté, ce qui a pu les conduire à fournir un jeu d'instructions et de fonctions abondant, en regard du nombre assez faible de séances consacrées à cette structure. L'épreuve infirme donc le jugement implicite des enseignants, et la comparaison entre classes indique que l'abondance du jeu d'instruction ne facilite pas l'acquisition.

Nous avons mis en évidence chez les élèves qui ont intégré le rôle des différents arguments des fonctions, la présence de difficultés déjà repérées dans la classe où ont été passées les épreuves EPR1 et EPR2, en particulier celles qui conduisent à des erreurs de calcul des ordinaux, et les difficultés de coordination entre ordinaux et cardinaux.(paragraphes 4.1 et 4.2). De même, ces élèves rencontrent des difficultés avec la concaténation, puisque au problème **BONJROU**, 65% des élèves forment la chaîne résultat par la concaténation de `A`, `B` et `C`, mais seul 35% concatènent dans l'ordre permettant d'avoir le résultat souhaité. (paragraphe 4.2.2). Nous avons souligné que les exercices proposés par les professeurs à ces classes dans le cadre de l'option sollicitent assez peu ces aspects, pour nous importants, de la structure de chaîne (paragraphe 5).

Le problème **RUOJNOB** devait permettre de mettre en évidence ou de confirmer des difficultés ayant trait à la construction d'itération dans le contexte d'utilisation des chaînes de caractères. De nombreux élèves (50%) traitent ce problème comme une question de cours sur le retournement d'une chaîne de caractères. Le quart des élèves seulement traite effectivement la question. Parmi ces élèves, on trouve très majoritairement les difficultés déjà rencontrées dans un cadre analogue : construction non itérative ; non prise en compte des états successifs.

Il semble (paragraphe 5) que le niveau scolaire, en particulier dans l'enseignement scientifique joue un rôle dans la réussite à l'épreuve ; l'orientation en Première S n'entraîne cependant nullement de façon automatique la réussite à l'épreuve.

Les différences entre résultats d'une classe à l'autre incitent à rechercher des facteurs de réussite ayant trait aux conditions d'apprentissage. Parmi ces conditions, le contenu des cours, et les énoncés des exercices proposés dans les classes permettent d'éclairer les performances d'ensemble assez faibles sur les trois classes ; ils ne permettent pas d'expliquer les différences de performances entre classes pour des élèves ayant des décisions d'orientation similaires ; seul un jeu d'instructions un peu moins abondant dans la classe F est à mettre en relation avec de meilleures performances pour les deux types d'orientation.

Les données présentes ne permettent pas d'en dire davantage sur les conditions d'apprentissages qui peuvent influencer les acquisitions ; au delà du strict énoncé du contenu des cours et exercices, la façon de présenter une structure, la tâche précise

de programmation confiée à l'élève au cours de la résolution des exercices, l'intervention de l'enseignant au cours de cette résolution, sont des éléments susceptibles de multiples variations ; notre étude indiquerait qu'on ne peut faire l'économie de l'analyse de ces facteurs, si l'on veut expliquer des différences d'acquisition.

Conclusions de la partie II

La partie II fait le bilan d'une étude expérimentale construite pour mettre à l'épreuve certaines hypothèses de la partie I:

- dans les représentations spontanées des objets informatiques et des éléments du langage qui s'y rapportent (affectation, notation fonctionnelle), c'est à dire dans les "plans" relatifs aux objets des **S.R.T.** que les sujets débutants mettent en oeuvre pour la résolution de problèmes de programmation, ces objets et écritures possèdent des propriétés déduites des représentations des objets dans le contexte familier,
- les "plans" relatifs aux traitements, que les débutants mettent en oeuvre pour des algorithmes itératifs, sont de la même façon influencés par les traitements dans le contexte familier. Les deux types de plans (relatifs aux objets, et relatifs aux traitements) sont interdépendants, c'est-à-dire que le débutant ne peut concevoir les éléments constitutifs d'un traitement, indépendamment des objets sur lequel il porte,
- la mise en évidence de ces "plans" et de leur interdépendance suppose un type de problème de programmation où un contexte familier (une situation connue) est mis en jeu, où le traitement par le dispositif ne peut être traduit directement de ce contexte, et où, par conséquent, la résolution impose la construction par le sujet d'un **S.R.T.** où objets et traitements se différencient de leurs homologues du contexte familier.

Plus précisément, cette partie, nous a permis d'examiner par une série d'observations ponctuelles en classe de Seconde option informatique comment, le choix d'un type de base étant fait, ces hypothèses se spécifient dans ce contexte, et de vérifier leur validité.

Le type de base choisi est le type "chaîne de caractères", qui fait lui-même intervenir les types ordinaux et cardinaux : nous définissons ce type par un jeu réduit de fonctions (`sous-chaîne`, `longueur`, `concaténation`, opérateurs de comparaison, fonctions de conversion de type) de façon à permettre la généralité de la démarche dans la classe des langages impératifs, et aussi parce que des problèmes conformes aux hypothèses ci-dessus peuvent s'appuyer sur des situations où la construction de fonctions sur les chaînes, composées des fonctions du jeu réduit apparaît comme une solution, alors que la donnée d'un ensemble étendu de fonctions ne permet pas à l'élève de structurer cet ensemble.

Nous avons, dans un premier temps, spécifié la première hypothèse en postulant l'existence de conceptions des objets déduites du contexte familier et non adaptées au traitement informatique, pouvant s'observer à travers quatre classes de difficultés :

- *D.Langage*: les difficultés liées à une conception naïve du langage de programmation (affectation, écritures fonctionnelles)
- *D.Sem.Cha*: les difficultés propres à la structure de chaîne
- *D.Ord.Card.*: les difficultés liées à la conception des ordinaux, et à la coordination ordinaux-cardinaux
- *D.Type* : les difficultés liées au typage des objets

Nous avons ensuite mené une observation sur 13 élèves d'une classe de Seconde Option Informatique, basée sur une épreuve sur papier et des entretiens. Elle confirme l'existence de ces difficultés, permet de mieux les comprendre, et montre que les problèmes construits selon les principes énoncés ci-dessus sollicitent effectivement ces difficultés.

- *D.Langage* se manifeste chez plus de la moitié des élèves : ces élèves n'emploient pas l'affectation des résultats intermédiaires comme décomposition du problème, et produisent donc des écritures fonctionnelles complexes, qui, pour eux, codent des actions sous une forme proche de métaphores en langage naturel, l'affectation étant présente seulement pour la correction syntaxique. Par contre, chez ceux qui réussissent, l'acquisition de l'affectation va de pair avec une meilleure compréhension des écritures, acquise, nous le pensons, grâce à la modularité permise par l'affectation des résultats intermédiaires. Nous soulignons que *D.Langage* ne saurait se concevoir indépendamment de la structure de chaîne, bien que les

éléments en jeu (affectation, écritures fonctionnelles) soient d'un point de vue conceptuel, indépendants du type des données sur lesquelles elles portent. Ceci confirme bien que, au niveau auquel les élèves raisonnent, ces éléments du langage sont conçus de façon dépendante des données sur lesquels ils portent.

- *D.Ord.Card.* se manifeste lors de la résolution d'un problème de construction d'une homothétie ordinale pour laquelle les élèves ne réinvestissent pas spontanément leurs connaissances en mathématiques dans le domaine des fonctions affines. De nombreuses difficultés concernant la coordination entre cardinaux et ordinaux se rencontrent également lorsque les élèves font intervenir dans leurs calculs le résultat de la fonction `longueur` qui peut s'interpréter à la fois comme ordinal et comme cardinal.
- Une même erreur d'écriture peut se trouver au confluent de plusieurs difficultés: `L` étant la longueur d'une chaîne donnée, on trouve par exemple chez de nombreux élèves, y compris parmi ceux qui n'ont pas rencontré les difficultés *D.Langage* la position `L-1` (au lieu de `L`) pour le dernier caractère: nous interprétons cette écriture comme un cas typique de difficulté de coordination entre cardinaux et ordinaux (*D.Ord.Card.)*, mais il faut considérer également que les élèves qui l'écrivent peuvent également comprendre cette écriture comme codant l'action "séparer le dernier caractère", qui est à mettre en relation avec *D.Langage* puisqu'une écriture fonctionnelle est comprise comme codant une action. *D.Sem.Cha.* est également en cause, puisque l'erreur apparaît seulement dans l'écriture d'arguments de la fonction `sous-chaîne` dont le rôle n'est pas encore bien précis pour les élèves.
- *D.Type* se manifeste par le choix du type d'une variable déterminé par la nature intuitive des données et non par le jeu de fonctions que les élèves font opérer sur cette variable. Elle se manifeste également dans l'écriture des constantes.
- Les erreurs de type peuvent également s'analyser comme résultant de la conjonction de plusieurs difficultés: lorsqu'un élève écrit:
  `LET RESULTAT$= MID$(ALPHABET$,DONNEE$+1,1)`, il fait une erreur de type, mais celle-ci est à relier à une conception des chaînes où les ordinaux ne sont pas distingués des caractères qu'ils pointent (*D.Ord.Card.*). *D.Sem.Cha.* est également concernée, puisque l'écriture intervient comme argument d'un fonction du type chaîne.

Nous avons ensuite examiné l'intervention au sein du **S.R.T.** de l'élève, de plans relatifs aux traitements, en conjonction avec les plans relatifs au données étudiés au chapitre précédent. Nous avons fait le choix de l'itération, en tant que construction algorithmique mettant à contribution les plans relatifs aux traitements, et d'une classe de problèmes définie à partir des principes énoncés ci-dessus: la recherche itérative d'informations dans une chaîne de caractères (problèmes de "*recherche-translation*"). Nous nous sommes appuyés comme ci-dessus, sur une observation de type clinique, basée sur une épreuve sur papier et un entretien, dont nous rappelons les résultats ci-dessous:

- Les réponses des élèves s'analysent comme des points d'équilibre résultant d'une conjonction de plans relatifs aux objets, observés au chapitre précédent, et de plans relatifs aux traitements souvent influencés par une conception naïve du traitement de données textuelles. Par exemple, la confusion (*D.Ord.Card.)* des ordinaux avec les caractères qu'ils pointent dans une chaîne s'observe en conjonction avec une conception du dispositif comme capable d'accéder de façon directe (ou associative) à la position d'une sous-chaîne donnée dans une chaîne. De même, la conception naïve du langage (*D.Langage*) s'observe en conjonction avec la non-intégration du rôle des éléments de l'itération (avertisseurs d'itération, compteur...).
- L'interaction entre conceptions des objets et conceptions des traitements se manifeste également lorsque la nécessité de coordonner un traitement itératif et un traitement alternatif fait ressortir *D.Langage* .
- Les problèmes de "recherche-translation" permettent effectivement la mise en évidence de cette interaction.
- L'influence des formulations en langage naturel sur les conceptions, semble moins décisive que ce que nous pensions au départ.

L'étude confirme donc que, bien qu'objets et traitements soient indépendants au niveau conceptuel, les **S.R.T.** que les élèves mettent en oeuvre pour la résolution de problèmes, font intervenir de façon liée les conceptions des objets et les conceptions des traitements; il en résulte que l'acquisition de capacités relatives aux traitements complexes (itération...) ne saurait s'opérer sans articulation avec les capacités relatives aux objets. Nous en avons vu un exemple avec des élèves qui, abandonnant une conception erronée des ordinaux au profit d'une conception plus adaptée, sont conduits à remettre en cause leur conception de l'accès aux parties d'une chaîne, et donc à construire une itération.

Nous avons ensuite mené une étude quantitative, par l'observation d'élèves de plusieurs classes, et de niveaux scolaires variés.

- Pour une bonne moitié des élèves, les fonctions qui définissent les chaînes, ainsi que l'utilisation sur des chaînes des éléments de base du langage, ne sont pas compris: beaucoup de réponses consistent en fait à "donner à l'ordinateur" toutes les informations sur le résultat souhaité, sans concevoir le traitement comme un calcul sur ces informations.
- Chez les autres élèves les difficultés repérées ci-dessus se rencontrent. Elles sont plus fréquentes dans les calculs qui sollicitent la structure de chaîne dans sa généralité (par exemple une sous-chaîne qui ne soit ni le début ni la fin d'une chaîne). En effet, dans des cas particuliers, la réponse correcte peut se révéler compatible avec les conceptions erronées.
- Les élèves qui manifestent un minimum de compréhension de l'utilisation de la structure de chaîne sont quasiment tous orientés en Première Scientifique, ce que nous avons retenu comme critère d'une scolarité réussie dans le domaine scientifique. Mais la réciproque n'est pas vraie, en ce sens que près de la moitié des élèves orientés en Première Scientifique échoue.
- La comparaison entre classes montre des disparités importantes, y compris si l'on considère des groupes d'élèves de même niveau scolaire. Seul l'emploi d'un jeu d'instructions moins abondant dans la classe qui réussit le mieux, peut être mis en relation avec ces disparités : dans l'ensemble, les professeurs ont fourni le jeu complet d'instructions présent dans le langage (qui se trouve être moins abondant dans le langage utilisé par la classe qui réussit le mieux), et traité des exercices assez classiques, mais qui, selon nous, ne sollicitent pas suffisamment la structure de chaîne.

Schématiquement, cette partie II vise à un constat des **S.R.T.** des élèves à un moment donné de leurs apprentissages et la partie III aura pour but d'examiner comment ces **S.R.T.** évoluent au cours des apprentissages. Néanmoins, sur le plan pédagogique, les observations rapportées ci-dessus permettent d'envisager des premières conséquences des conceptions que nous avançons dans cette thèse : l'intervention pédagogique auprès d'élèves en situation de résolution, lorsqu'elle ne peut s'appuyer sur la connaissance précise des difficultés de l'élève, en relation avec ses conceptions, semble porter tendanciellement sur les aspects syntaxiques, et l'interprétation de résultats d'exécution. Une intervention de ce type peut réussir auprès d'élèves disposant de conceptions adéquates des éléments généraux du langage : ils peuvent ainsi constituer un **S.R.T.** correspondant à une situation nouvelle, au sein de conceptions existantes. Par contre, d'autres élèves dont les conceptions sont très éloignées du fonctionnement du dispositif, n'obtiennent pas de meilleure compréhension en améliorant leur syntaxe, et ne peuvent interpréter les résultats d'exécution. Nous pourrons donc nous appuyer sur ces résultats pour l'essai d'ingénierie didactique de la partie III.

Partie III: Ingénierie didactique

Une progression à partir de séries de problèmes utilisant les chaînes de caractères

Chapitre 9: Les problèmes et la situation pédagogique

Chapitre 10: Expérimentation auprès d'élèves de Seconde Option Informatique

Chapitre 9 Les problèmes et la situation pédagogique

Classification des tâches
Principes de conception des problèmes et de la situation de résolution.
Analyse a priori des énoncés.

Dans les chapitres de la partie **II**, nous avons mis en évidence des difficultés assez largement répandues chez des élèves de seconde, concernant l'acquisition des chaînes de caractères, en lien avec les conceptions qu'ils mettent à contribution lors de la résolution de problèmes de programmation: les difficultés *D.Langage* liées à une conception naïve du langage de programmation (affectation, écritures fonctionnelles), les difficultés *D.Sem.Cha* propres à la structure de chaîne, les difficultés *D.Ord.Card* liées à la conception des ordinaux, et à la coordination ordinaux-cardinaux, les difficultés *D.Type* liées au typage des objets. Cette mise en évidence vient confirmer les hypothèses énoncées en partie **I** sur la nécessaire constitution, chez les débutants, de **S.R.T.** articulant des conceptions adaptées des objets, et des conceptions des traitements se dégageant progressivement des particularités des objets, processus qui, chez de nombreux élèves, ne semble pas achevé à l'issue de la classe de Seconde.

En partie **I**, également, nous avons émis des hypothèses sur le type de problème pouvant conduire à cette constitution: d'une part le contexte du problème doit permettre à l'élève de donner une signification aux objets et traitements, fournir un(des) plan(s) guidant sa résolution; d'autre part , au cours de la résolution, l'élève doit rencontrer les limites de conceptions des objets et des traitements issues de ce contexte: ces problèmes doivent donc, à la différence de problèmes où les données sont "déjà codifiées", conduire le sujet débutant à opérer des différenciations par rapport à ses conceptions des objets et traitements intuitifs et doivent donc faire intervenir une "codification" d'objets ou de données intuitifs sous forme de données informatiques et la production d'une solution comme "calcul" sur ces données codifiées. Ces problèmes doivent faire interagir les concepts informatiques de base et au contraire, éviter de faire intervenir des éléments conceptuels en dehors du champ de l'informatique. Ayant construit les items des 3 épreuves de la partie **II**, en fonction de ces impératifs, nous pensons confirmée l'hypothèse rappelée ci-dessus Une autre hypothèse est que l'existence chez un élève de conceptions encore trop peu différenciées par rapport aux conceptions naïves, ne doit pas être comprise comme un obstacle empêchant un enseignement de l'informatique; au contraire un élève ayant rencontré et surmonté les difficultés qui en résultent aura, selon cette hypothèse, gagné des représentations mentales plus opératoires sur lesquelles il pourra appuyer ses acquisitions ultérieures. Un corollaire de cette hypothèse est donc la possibilité d'établir des séries de problèmes du type décrit ci-dessus et organisées en progression visant à faire évoluer les représentations des élèves.

Pour tester ces hypothèses, nous avons donc construit des séries de problèmes visant à travailler sur ces difficultés ; ces problèmes sont de présentation et de difficulté analogues aux problèmes dont nous avons donné l'énoncé dans la partie II : basés sur une modélisation simple de situations connues des élèves, il conduisent les élèves :

- à construire des fonctions d'accès à l'aide de la fonction sous-chaîne et de la concaténation
- à considérer des nombres aussi bien comme chaînes de chiffres que pour leur valeur numérique,
- à construire des itérations sur les chaînes pour résoudre des problèmes de comptage d'occurrences, et de "*recherche-translation*",

Ces problèmes s'adressent à des élèves pour lesquels il s'agit de la première prise de contact avec la structure de chaîne de caractères. Ils ont fait l'objet d'une pré-expérimentation dans une classe de Seconde option informatique, au cours de

l'année scolaire 87-88; à la suite de cette pré-expérimentation, les énoncés ont été modifiés pour faire davantage de place à l'opération de concaténation. Les séries modifiées ont fait l'objet d'une expérimentation également dans une classe de Seconde option informatique, mais au cours de l'année scolaire 88-89; cette expérimentation a permis une observation précise de deux groupes d'élèves, qui est rapportée et analysée au chapitre **10**: le but de l'expérimentation est de valider les séries de problèmes, en rapportant les difficultés rencontrées par les élèves, la façon dont ils ont progressé en les résolvant, et de vérifier les acquisitions par une épreuve sur papier. Nous voulons ainsi examiner dans quelle mesure il est possible pour les élèves de surmonter les difficultés d'acquisition , et vérifier qu'un type de donnée comme les chaînes de caractères peut être le support pour des acquisitions de portée générale en informatique: calculs d'ordinaux, itération sur un ensemble non numérique, notion de type, et commencer à répondre aux questions posées au chapitre **4** concernant la situation pédagogique de résolution dirigée.

Dans ce chapitre, nous donnons tout d'abord une **classification des tâches** qui peuvent être demandées aux élèves dans la résolution de ces problèmes, en lien avec les représentations sur lesquelles nous voulons agir, puis nous donnons quelques indications sur le **contexte didactique** dans lequel les problèmes présentés dans ce chapitre se conçoivent: exigence du **calcul d'un résultat**, préalable à l'instruction de sortie, choix du **travail dans le langage de programmation** (c'est-à-dire expression des problèmes et des solutions dans le langage réellement exécutable utilisé par les élèves) au cours de l'apprentissage et pour l'évaluation des acquisitions, plutôt que l'exigence d'une **analyse** (comprise comme expression d'un procédé de résolution préalable à la programmation). Nous présentons enfin une **analyse a priori** de chacun des énoncés (tâche de l'élève, difficultés attendues, indications de solutions..).

1. Une classification des tâches dans des problèmes sur les chaînes de caractères

L'étude des cours et exercices, dans les classes où les observations ont été menées, nous a conduit à penser que la structure de chaîne est sollicitée seulement dans des aspects partiels (par exemple, les sous-chaînes peuvent être calculée seulement à partir du rang 1) et, par conséquent, dans les exemples et exercices, les réponses exactes peuvent parfaitement être compatibles avec les conceptions erronées. Par ailleurs, certains problèmes, particulièrement ceux traités en fin d'apprentissage (conversion décimale binaire, jeu du pendu...), nous ont semblé accumuler les difficultés, sans nécessairement solliciter davantage la structure de chaîne; la compréhension du langage est bien sûr fortement sollicitée dans ces exemples, mais l'accumulation des difficultés a pour conséquence un manque total d'autonomie des élèves: comme il n'y a pas résolution par l'élève, il ne peut y avoir évolution des conceptions. Une classification des tâches demandées aux élèves dans un problème donné doit nous permettre de vérifier d'une part que la structure de chaîne, et particulièrement les difficultés mises en évidence en partie **II** sont réellement sollicitées, et d'autre part que la tâche reste effectivement à la portée des élèves.

Nous définissons ci-dessous quatre types de tâches, en fonction de l'analyse de la structure de chaîne du chapitre **5** et en relation avec les difficultés mises en évidence en partie **II**. Nous repérons par un libellé en italiques majuscules chacun de ces types; chaque problème pourra bien entendu correspondre à plusieurs types de tâches.

- type (*CALCUL ARGUMENTS*): ce type concerne les problèmes les plus simples: les élèves ont seulement à compléter une partie des arguments des fonctions; on leur demande de comprendre la structuration des données, et de résoudre les questions de mise en oeuvre précise des fonctions (écritures fonctionnelles, calculs d'ordinaux et de cardinaux).
- type (*CONSTRUCTION DE FONCTIONS*): dans ces problèmes, la structuration des données peut être plus complexe: une représentation de cette structure à l'aide des chaînes de caractères est proposée dans l'énoncé; les élèves ont à construire les fonctions sur ces données; ils ne disposent pas à ce niveau, de la possibilité de construire des fonctions nommées par un identificateur, et doivent donc recourir à

l'affectation et ceci d'autant plus qu'ils sont moins à l'aise avec la notation fonctionnelle; l'acquisition de l'affectation et d'autre part, la compréhension des fonctions sur les chaînes et leur composition. sont également également en jeu dans les tâches de ce type.

- type (*CONSTRUCTION ITERATION*): nous souhaitons, à travers ce type de tâche, solliciter la coordination entre représentations des objets et représentations des traitements non séquentiels, tout en restant à un niveau de complexité compatible avec une activité de résolution autonome des élèves: les parcours itératifs envisagés présentent donc un seul point de sortie, et ne nécessitent pas d'articulation complexe entre variables itératives: les élèves ont à construire le parcours d'une chaîne à l'aide d'un compteur et éventuellement d'autres variables itératives, dans le but d'y trouver une information[107].
- type (*CHOIX TYPE*): les élèves auront dans ces problèmes, à faire le choix d'utiliser une représentation des données, en fonction des opérations que l'on souhaite effectuer sur ces données (et donc d'une abstraction implicite), plutôt qu'en fonction de leur apparence externe; en pratique, il s'agira de données numériques pour lesquelles les élèves auront à choisir entre le type "chaîne" et un type numérique. La nécessité d'utiliser des fonctions de conversion apparaîtra également à cette occasion.

2. Principes conduisant à la conception de séries de problèmes et de la situation de résolution

2.1 Les énoncés mettent l'accent sur le calcul et l'affectation plutôt que sur l'affichage

L'analyse des structures générales des langages impératifs (chapitre 5) montre que l'affectation est, si l'on n'utilise pas la possibilité de définir des actions nommées, et si les instructions de lecture sont placées en début de programme, la seule véritable action sur le dispositif; nous avons montré que l'affectation se définit de façon non triviale, comme une action sur le dispositif, celui-ci étant défini comme l'ensemble des relations logiques entre variables du programme. Par contre, les instructions de sortie, si elles produisent des effets visibles extérieurement ne constituent pas des actions sur le dispositif ainsi défini[108]. Nous avons vu au chapitre 6 que l'acquisition d'une compréhension de l'affectation n'a rien d'immédiate: les élèves, dans leur majorité, prêtent aux écritures fonctionnelles et/ou aux instructions de sortie la capacité d'agir sur le dispositif. Ainsi, l'élève peut produire une écriture fonctionnelle de façon isolée, ou précédée d'une affectation à une variable qui n'est pas réutilisée dans la suite: dans ce cas, il n'y a pas interférence avec les instructions de sortie: c'est à l'écriture fonctionnelle seule qu'est prêtée la capacité d'action. Dans d'autres cas, une instruction d'affichage porte sur une écriture fonctionnelle, dans le but d'indiquer au dispositif de calculer un élément nécessaire à la suite du programme: par exemple l'élève écrira `afficher sous-chaîne(mot, 1,1)` pour calculer le

---

[107] Après avoir lu le texte des problèmes de ce type des séries 1 et 2 ci-dessous, on fera sans doute la remarque qu'une solution pourrait être construite sans itération, en utilisant la fonction `POSITION` : rappelons que la fonction `POSITION` prend deux arguments : `chaine1` et `chaine2` ; elle renvoie le rang du premier caractère de la première occurrence de `chaine2` comme sous-chaine de `chaine1` , ou `zéro`, s'il n'y a pas d'occurrence ; nous avons souligné plus haut que notre but n'est pas d'explorer l'acquisition d'un jeu complet d'instruction sur les chaînes de caractère, et que la fonction `POSITION` ne fait pas partie du jeu réduit d'instructions que nous avons proposé ; il nous semble en fait intéressant, concernant les rapports entre l'itération et la structure "chaîne de caractère" de construire une telle fonction en réponse à des problèmes (voir Chapitre 7).

[108] Sauf si l'on considère le dispositif de sortie (généralement l'écran texte ou graphique) comme un ensemble de variables implicites (pixels ou caractères déjà affichés, position du curseur, état du défilement de l'écran..). Une instruction de sortie est dans ce cas un ensemble d'affectations implicites à ces variables implicites, et l'on est ramené à la question de la compréhension de l'affectation, mais dans un contexte qui peut être plus obscur.

premier caractère de mot de façon à l'utiliser dans un calcul ultérieur, sans que l'affichage de ce caractère soit pertinent dans le problème posé[109].

Il existe des contextes didactiques qui "court-circuitent" le calcul et l'affectation par le recours à l'affichage

Il nous semble que la conception des problèmes proposés aux élèves peut interagir avec cette compréhension des instructions d'affichage et des écritures fonctionnelles comme agissant sur le dispositif, au détriment de l'affectation. Nous donnons deux exemples d'énoncés montrant comment, dans le cas de problèmes utilisant les chaînes de caractères, les particularités de l'affichage permettent un "raccourci didactique", évitant les difficultés de la concaténation et de l'affectation:

- Enoncé 1:"Un mot étant entré par l'utilisateur, donner le mot obtenu en inversant l'ordre des caractères".

  Le verbe "*donner* " est volontairement ambigu; il peut signifier afficher ou calculer. Interprétons selon la première signification: l'énoncé devient: "*afficher le mot obtenu en inversant l'ordre des caractères*", ce qui se simplifie en "*afficher le mot dans l'ordre inverse des caractères*"; on obtient par exemple l'algorithme suivant:

```
lire MOT
I ← longueur(MOT)
tant que I > 0
   afficher sous-chaîne(MOT,I,1) sans passer à la ligne
   décrémenter I
qt
```

  Si, par contre, on interprète "*donner*" comme l'indication que le mot dans l'ordre inverse des lettres est un résultat qui doit pouvoir être éventuellement affiché, mais surtout pouvoir être réutilisé, il convient d'utiliser une variable itérative (RESULTAT), et la concaténation:

```
lire MOT
I ← longueur(MOT)
RESULTAT ← chaine-vide
tant que I > 0
   LETTRE ← sous-chaîne(MOT,I,1)
   RESULTAT ← concaténation(RESULTAT,LETTRE)
   décrémenter I
qt
```

  La première interprétation de l'énoncé fait l'économie des principales difficultés de ce problème:

  - une difficulté concernant l'utilisation de la concaténation: la forme concaténation(RESULTAT,LETTRE) n'est pas celle qui vient spontanément à l'esprit; [ARSAC 83] rapporte que des étudiants de niveau universitaire considèrent spontanément cette forme comme erronée.
  - des difficultés concernant l'itération: emploi d'une variable itérative, initialisation...

Pour profiter de cette économie, les énoncés portant sur les chaînes de caractères, proposés aux élèves en phase d'alphabétisation peuvent privilégier la compréhension des résultats comme "valeurs à afficher"; ces énoncés conduisent assez facilement les élèves à des programmes qui produisent des effets sur le périphérique d'affichage; ils se justifient sans doute ainsi par une certaine satisfaction des élèves.

- Enoncé 2: «*éditer un mot fourni par l'utilisateur, lettre par lettre, de haut en bas*». cet énoncé est tiré d'un document de réflexion d'enseignants de l'option informatique

---

[109] Ce type de difficulté peut sembler très liée à la classe de langage impérative. Des observations (non publiées) sur des étudiants, futurs enseignants en mathématiques, programmant en LOGO nous ont permis de constater l'impossibilité pour certains, de distinguer une fonction rendant une valeur, de la procédure affichant à l'écran cette même valeur. Il y a donc bien dans les deux "paradigmes de programmation" , non compréhension du mode d'évolution de l'état du dispositif (affectation en impératif, résultat rendu par une fonction en fonctionnel), et transfert à l'instruction de sortie de ces capacités d'évolution.

[CURFIP-USTLFA 89][110]

Si l'on désire que le programme produise une valeur pouvant être réutilisée, il convient de considérer la chaîne constituée des caractères spéciaux "Retour Chariot" et "Saut de ligne", dont l'affichage produit le passage au début de la ligne suivante. Ces caractères se calculent à l'aide d'une fonction de conversion (fonction CHR$ en BASIC, CHR en PASCAL, CAR en LSE), connaissant leur code interne dépendant de la machine. Cette chaîne étant supposée calculée et affectée à la variable chaîne PASSAGE_A_LA_LIGNE , une solution s'écrirait:

```
lire MOT
L ← longueur(MOT)
RESULTAT ← chaine-vide
pour I ← 1 jusqu'à L
 LETTRE ← sous-chaîne(MOT,I,1)
 RESULTAT ← concaténation(RESULTAT , LETTRE, PASSAGE_A_LALIGNE)
fin pour
```

Le contexte didactique conduit à penser que cette solution n'est pas celle qui est attendue des élèves; en effet l'utilisation des caractères spéciaux suppose une certaine connaissance de l'informatique et de ses matériels; la solution attendue est clairement du type: "répétition d'affichages sans calcul".

```
lire MOT
L ← longueur(MOT)
pour I ← 1 jusqu'à L
   afficher sous-chaîne(MOT,I,1) et passer à la ligne
fin pour
```

L'impasse qui peut résulter du recours à l'affichage: une observation en classe

Il existe donc bien des énoncés (et par conséquent une pratique didactique) visant à faire l'économie de difficultés en programmation en privilégiant l'aspect "affichage" sur l'aspect "calcul-affectation". Rapportons une observation qui montre l'impasse à laquelle peut conduire une telle pratique:

Il s'agit d'une classe de Seconde Option Informatique, en fin d'année scolaire; cette classe est jugée plutôt bonne par le professeur. Les élèves ont à "*programmer le jeu du pendu*", c'est-à-dire que l'ordinateur doit conduire l'utilisateur à deviner progressivement les lettres d'un mot de la langue française; dans un premier module, l'ordinateur doit afficher une chaîne dont la première lettre et la dernière sont les mêmes que le mot à deviner; cette chaîne est de même longueur que le mot à deviner, et les caractères autres que la première et dernière lettre sont de tirets. Ensuite, l'utilisateur doit entrer une lettre; si cette lettre appartient au mot à deviner, elle remplace dans la chaîne le tiret correspondant à sa place, et ainsi de suite.

La première séance (1h30) se déroule ainsi:

- les élèves rédigent le premier module sous forme d'un affichage, et obtiennent rapidement un résultat satisfaisant:

```
lire MOT
L ← longueur(MOT)
afficher sous-chaîne(MOT,1,1) sans passer à la ligne
pour I ← 2 jusqu'à L-1
   afficher un tiret sans passer à la ligne
fin pour
afficher sous-chaîne(MOT,L,1) et passer à la ligne
```

puis, avec davantage de difficultés, ils cherchent si la lettre entrée par l'utilisateur appartient au mot, et calculent sa position dans le mot. Il se produit ensuite un blocage général de la recherche des élèves; en effet, pour la première lettre trouvée par l'utilisateur, l'affichage est encore possible, bien que lourd: supposons que POSI représente la position de la lettre trouvée (elle-même affectée à la variable chaîne LETTRE); on obtient:

```
lire MOT
L ← longueur(MOT)
afficher sous-chaîne(MOT,1,1) sans passer à la ligne
```

---

[110] ("*compte-rendu des travaux des enseignants de l'O.I.* "CURFIP-USTLFA vol 2 88/89)

```
pour I ← 2 jusqu'à POSI-1
  afficher un tiret sans passer à la ligne
fin pour
  afficher LETTRE sans passer à la ligne
pour I ← POSI+1 jusqu'à L-1
  afficher un tiret sans passer à la ligne
fin pour
afficher sous-chaîne(MOT,I,L) et passer à la ligne
```

Mais, dès la seconde lettre, ce mode de résolution devient trop compliqué à mettre en oeuvre.

Si les élèves avaient *calculé* la chaîne résultat à la première question, il leur aurait été aisé de calculer ensuite la chaîne obtenue en remplaçant le tiret par la lettre à sa place dans le mot[111] Il nous a semblé que le professeur n'avait pas envisagé cette difficulté en présentant le problème; à la séance suivante (la dernière de l'année), il s'est trouvé contraint d'abandonner la résolution du problème en l'état; les élèves n'acceptaient pas en effet de recommencer le premier module pour lequel ils jugaient leur solution satisfaisante. Le contrat didactique implicite consiste à considérer comme valable l'algorithme qui conduit *à un affichage sur l'écran*, indépendamment du fait qu'un *résultat* ait été ou non calculé; nous avons vu que ce contrat permet, en économisant des difficultés concernant l'itération, d'obtenir plus vite des programmes satisfaisants pour les élèves; il présente les inconvénients suivants:

- il entretient la confusion notée plus haut entre le *calcul* d'un résultat, et son *affichage* sur le périphérique de sortie.
- le programme est considéré du point de vue des *actions* (les affichages) et non du point de vue des *situations* (les valeurs des variables).
- concernant les chaînes de caractères, ce contrat *évacue la concaténation*, et empêche de considérer les "manipulations sur les chaînes" comme un véritable calcul.

C'est pourquoi nous établirons, dans ces séries de problèmes l'exigence du calcul d'un résultat avant son affichage: les valeurs nécessaires au problème (et en particulier le résultat) étant *calculées*, elles seront, en programmation impérative, *affectées* à une variable (en programmation fonctionnelle, on construirait une fonction); l'affichage n'intervient que de façon secondaire, les valeurs ayant été calculées. Cette exigence se justifie auprès des élèves par le fait que tout programme doit être conçu pour être utilisé comme programme élémentaire d'un programme plus vaste, et par des raisons de clarté de la lecture.

2.2 Le travail dans le langage de programmation présente des limites mais paraît le seul choix adapté à des débutants

La mise en place d'une séquence didactique suppose de façon implicite ou explicite des hypothèses et des choix quant aux conditions dans lesquelles les élèves peuvent accéder aux acquisitions visées. Nous nous intéressons d'abord aux énoncés de problèmes comme variable didactique, mais il existe bien d'autres éléments composant une situation didactique en informatique, qui appellent des choix, et ont des conséquences sur les acquisitions; nous allons énoncer certains de ces choix, et pour les situer nous rappelons d'abord les indications données par le programme officiel de l'option informatique (présentées en chapitre 4), nous les discutons, puis nous énonçons nos propres choix pour les séances de résolution que nous mettons en place.

La place de l'analyse dans les instructions en vigueur dans l'option; discussion

Dans le contexte professionnel, et dans celui de l'enseignement universitaire,

---

[111] En PASCAL, langage utilisé dans cette classe le type chaîne est compatible avec le type tableau ; c'est-à dire qu'une chaîne de longueur L est un tableau de caractères indexé de 0 à L, l'élément d'index 0 représentant la longueur, les L éléments suivants représentant les caractères ; dans un langage n'offrant pas cette particularité, le remplacement d'un caractère dans une chaîne à une position donnée se fait de façon un peu plus lourde, mais sans grande difficulté ; on a également la possibilité de considérer le résultat comme un tableau, et non une chaîne, ce qui est plus cohérent avec les opérations qu'on souhaite effectuer.

l'informaticien se doit d'utiliser des concepts suffisamment généraux pour être indépendants d'un dispositif (matériel et logiciel) donné et de s'appuyer sur des méthodes de résolution aisément transférables. C'est pourquoi, dans ces contextes, l'analyse du problème, préalable à l'implémentation sur un dispositif donné, est fondamentale: spécification du problème comportant notamment l'abstraction des données, explicitation du procédé de traitement puis de l'algorithme pour un dispositif suffisamment général. En précisant «*On attachera donc la plus grande importance à l'analyse du problème, préalable à toute programmation*», le commentaire du programme de Seconde ([CNDP 1987]) transpose au niveau pédagogique cette démarche. Elle est systématisée par des méthodes d'enseignement pour débutants, comme la méthode MEDEE dont l'ouvrage de référence [DUCRIN 84] comprend un tome consacré à l'analyse (*du problème à l'algorithme*) et un autre consacré à l'implémentation dans un langage (*de l'algorithme au programme*).

Si une telle démarche était effectivement possible avec tous les élèves, les avantages que l'on pourrait en attendre seraient les suivants:

- l'élève, s'il trouve des réponses lui permettant de s'adapter au problème, dès le stade de l'analyse, transfère des connaissances à partir de la résolution de problèmes antérieurs, par exemple ceux dont la résolution a été *montrée* par les professeur. Il manifeste ainsi une représentation du dispositif suffisamment générale pour être indépendante du domaine de tâche.
- l'élève doit justifier par le raisonnement le fonctionnement correct de son programme, sans le support des résultats d'exécution. Ainsi, pour valider son analyse, l'élève s'appuie sur des représentations mentales fortes et construites, qui lui permettent d'anticiper les réponses du dispositif y compris dans des domaines de tâche nouveaux.[112]
- la phase d'implémentation de la solution sur un dispositif donné permet de conforter cette acquisition conceptuelle plus précisément, l'autorisation donnée par l'enseignant de passer au travail sur machine, ainsi que l'utilisation de codes existants dans le langage de programmation pour concrétiser les structures que l'élève a construites, contribuent à l'institutionnalisation, qui permet de donner aux productions de l'élève leur double statut de savoir scientifique et de production effective. Les représentations fortes et construites s'institutionnalisent en concepts.

Nos observations de débutants en phase d'alphabétisation en partie II font douter de la possibilité de mettre en oeuvre un tel schéma de résolution auprès de la majorité des élèves. Les élèves disposent de conceptions dont on peut penser qu'elles sont généralement assez prégnantes: de ce fait, elles leur permettent de produire des anticipations des réponses du dispositif; mais elles sont trop locales, trop liées à une expérience donnée pour pouvoir servir dans un domaine présentant un peu de nouveauté. En fait, dans la majorité des cas, c'est l'expérience intuitive qui fonde les conceptions face à une tâche nouvelle, et ce d'autant plus que l'élève n'a pas été confronté aux contraintes du dispositif. C'est pourquoi les réponses anticipées sont généralement fausses. Il faudrait, pour qu'il en aille autrement, que les contraintes du dispositif aient été suffisamment intégrées pour que, par une intériorisation de son dialogue avec le dispositif, l'élève aperçoive les grandes lignes de ce que serait une réponse du dispositif; c'est ainsi que procède un programmeur confirmé et il est possible que des élèves plus avancés puissent procéder ainsi.

---

[112] En distinguant un raisonnement *sans le support des résultats d'exécution* d'un raisonnement qui prendrait en compte ces résultats, nous ne voulons pas dire que les résultats d'exécution peuvent apporter une preuve quelconque de validité. Il nous semble que les résultats d'exécution peuvent apporter une aide au raisonnement, un peu comme le fait la figure dans le raisonnement en géométrie: ainsi, si l'on veut démontrer à un niveau élémentaire, que la composée de deux symétries par rapport à deux points distincts est une translation, on pourra difficilement se passer d'une figure (tracée effectivement, ou mentale) faisant apparaître un triangle et les milieux de ses côtés. L'exécution du programme est un peu analogue au tracé de la figure : sans rien prouver, il peut guider le raisonnement, faire apparaître des schémas déjà rencontrés. La capacité à anticiper, dans une situation nouvelle, le résultat d'exécution est l'analogue de la capacité à se représenter mentalement une figure : elle permet un raisonnement d'emblée plus général, mais suppose une représentation mentale structurée du domaine.

Le programme officiel semble d'ailleurs prévoir la nécessité d'une confrontation aux contraintes du dispositif, préalable à l'activité de résolution, si l'on peut interpréter ainsi l'indication pédagogique: «*...le professeur développera des exemples typiques, en insistant sur les aspects méthodologiques. En travaux pratiques, les élèves pourront reprendre ces exemples et traiter de façon analogue, d'autres problèmes de difficulté comparable*». Cette activité du professeur indiquée dans le programme pourrait avoir pour but de montrer comment une solution se construit en prenant en compte les contraintes du dispositif: ces contraintes seraient mises en évidence par le professeur au fur et à mesure du processus de résolution. Cette prise en compte des contraintes du dispositif au seul niveau du discours de l'enseignant nous semble tout à fait insuffisante pour une réelle intégration de ces contraintes par le sujet. Pour illustrer cette question, et montrer qu'elle ne se pose peut-être pas seulement au niveau de l'alphabétisation, nous rapportons une expérience personnelle:
Il y a plusieurs années je participais à un atelier d'informaticiens autour de la question "Comment aborder la résolution d'un problème en informatique ?". Ma connaissance de l'informatique se réduisait alors à la programmation d'algorithmes de calculs approchés en mathématiques. Le groupe des participants examinait plusieurs approches du problème consistant à programmer l'ordinateur pour qu'il donne des solutions au jeu connu sous le nom de "Le compte est bon"; dans ce jeu, 7 nombres à un ou deux chiffres, et un nombre à 3 chiffres sont tirés au hasard; il s'agit d'exhiber un calcul sur les nombres à un ou deux chiffres, dont le résultat approche le plus prés possible le nombre à 3 chiffres. Le groupe débattait de la question centrale de ce problème: comment générer la suite des calculs possibles à l'aide des 7 nombres donnés; diverses représentations de ces calculs étaient envisagées; les générations récursives et itératives étaient comparées. J'étais très mal à l'aise au cours de cette réunion, car, pour moi, il n'y avait tout simplement pas de problème: un ordinateur étant capable de faire des calculs très rapidement, il suffisait de les lui faire exécuter, et de s'arrêter quand le résultat serait suffisamment proche du nombre à trois chiffres; bien qu'étant déjà un peu familier de la programmation, un type de contrainte des dispositifs informatiques (la nécessité d'une représentation du calcul) m'était totalement étranger car lié à un domaine nouveau: celui de la résolution de problèmes par la machine; il m'était par conséquent impossible d'entrer dans la problématique du débat.

Il semble clair que:

- les contraintes du dispositif impliquées par les notions d'un domaine, font partie elles-même de ce domaine, c'est-à-dire que l'intégration de ces contraintes ne peut se faire sans un début de conceptualisation dans le domaine; ceci s'observe particulièrement dans le domaine des premiers apprentissages, mais il se peut que la question se pose également à chaque acquisition dans un domaine nouveau, y compris pour un programmeur confirmé.
- ce début de conceptualisation suppose une activité autonome de résolution par le sujet en phase d'apprentissage, pour intégrer ces contraintes dans son propre système de représentations. La considération de la façon dont un *expert*, qui dispose par définition de représentations élaborées, prend en compte les contraintes dans son activité de résolution est, par contre, à elle seule, un apport insuffisant.

Une illustration des difficultés auxquelles se heurte le schéma pédagogique décrit par le programme de l'option est l'existence de "perversions" couramment rencontrées, et explicitement dénoncées par le programme: «*on évitera qu'elle* (l'analyse) *ne soit considérée que comme une paraphrase en français courant du programme à écrire* (...)*elle doit évidemment précéder la programmation, et non pas la suivre*». Ces "perversions", semblent assez courantes à tous les niveaux et témoignent que la phase d'analyse peut-être considérée par les élèves comme une exigence didactique formelle, sans rapport avec les nécessités de la résolution du problème. Nous avons constaté, au chapitre **7**, dans le cas d'un problème de "recherche-translation", impliquant de plus la coordination entre une alternative et une itération (problème **CRYPTAGE**), que l'analyse demandée aux élèves préalablement à leur programme ne semble pas avoir d'influence sur leur résolution: s'ils pensent que la formulation de l'énoncé est directement compatible avec le langage

du dispositif, alors leur analyse est une paraphrase de l'énoncé; dans le cas contraire, l'analyse est isomorphe au programme qu'ils écrivent (par exemple l'alternative est oubliée à la fois dans l'analyse et le programme).

Le choix du travail avec le langage de programmation et ses limites

Les schémas pédagogiques visant à ce que les élèves résolvent et raisonnent au niveau d'un dispositif plus général que le langage réellement disponible, et en dehors du "feed back" d'un dispositif réellement présent, nous paraissent donc inadaptés pour l'alphabétisation. Il nous semble par ailleurs que, si en ce qui concerne les énoncés de problèmes nous avons les moyens d'innover, la construction de situations pédagogiques réellement opératoires est actuellement hors de portée. C'est pourquoi nous devons nous contenter d'approfondir la compréhension de situations existantes: comme nous l'avons indiqué au chapitre 4, nous souhaitons en priorité approfondir l'étude des situations de résolution par petits groupes, encadrées par l'enseignant, l'ordinateur étant à disposition pour des essais de compilation et/ou d'exécution; ces conditions sont celles que les élèves que nous comptons observer connaissent habituellement avec leur professeur et que connaissent beaucoup d'élèves des classes d'option informatique: le dispositif est présent physiquement; les élèves peuvent en obtenir des réponses au cours de l'élaboration de leur solution (messages d'erreur, résultats d'exécution...). La solution requise des élèves est un programme exécutable; sa rédaction doit permettre à l'élève de justifier au moins oralement, l'adéquation du programme aux exigences de l'énoncé du problème.

L'analyse des séances de résolution montrera dans quelle mesure cette situation permet que les élèves se heurtent à des difficultés conceptuelles, et les résolvent. Il est clair, cependant, que ce ce type de situation ne peut être conçu comme totalement satisfaisant: il est tout à fait probable que les représentations qui en résultent chez les élèves n'ont pas le caractère de généralité qu'ils auraient eu dans le schéma pédagogique prévu dans le programme de l'option, si ce schéma avait été possible. Nous nous trouvons en fait devant l'alternative décrite ainsi par [BROUSSEAU 86]: «*...faute de situations suffisamment accessibles, suffisamment efficaces et en nombre suffisamment petit pour permettre à des élèves d'âge quelconque d'accéder d'emblée, par adaptation, à une forme de savoir que l'on puisse considérer comme correcte et définitive, le professeur a le choix entre enseigner un savoir formel et dénué de sens ou enseigner un savoir plus ou moins faux qu'il faudra rectifier*».

Nous avons donc fait le choix d'une situation conduisant à un savoir *plus ou moins faux*, présentant en tout cas un caractère limité: les énoncés que nous proposons ont été conçus en fonction de difficultés liées à la structure de chaîne et aux représentations, et non aux particularités du langage de programmation employé par les élèves; si ce sont effectivement ces difficultés que les élèves résolvent, nous n'aurons pas introduit des savoirs qui pourraient gêner des acquisitions ultérieures, indépendantes d'un langage ou tout au moins d'une classe de langages. La question de la façon dont ce savoir demande à être, par la suite, *rectifié*, est totalement ouverte.

La validation des acquisitions

Nous prendrons comme indice d'acquisition la production d'un programme sans la présence du dispositif mais exprimé dans le langage de programmation, pour la résolution d'un problème représentatif des difficultés en cause; concrètement, la résolution des séries d'exercices sera suivie d'une épreuve sur papier. De réelles capacités seront ainsi repérées témoignant d'une différenciation des représentations, et d'une analyse au moins implicite du problème. Le travail demandé aux élèves s'exprimant dans un langage de programmation (ici BASIC), on pourra se demander si les capacités repérées dans ces conditions ne sont pas dépendantes du langages. L'élève saura-t'il les transférer pour la programmation dans un autre langage, ou pour une analyse indépendante d'un langage? Comme pour les situations pédagogiques, nous pouvons seulement dire que nous n'introduisons pas de savoirs qui seraient valables seulement dans le langage; pour le reste, ces questions sont ouvertes.

2.3 Les énoncés de problèmes peuvent être adaptés à différents langages

L'utilisation, avec les élèves d'un langage algorithmique pour l'expression des énoncés et des solutions ne paraissant pas pertinent pour les premières acquisitions, les élèves disposent d'un seul langage pour coder algorithmes et programmes. C'est pourquoi nous nous proposons d'utiliser ce langage de programmation dans les énoncés qui nécessitent l'expression de tout ou partie d'un programme. En cohérence avec nos propositions du chapitre **5**, nous employons avec les élèves seulement un jeu restreint d'instructions, commun à la plupart des langages où la structure de chaîne est implantée. Le libellé des énoncés tel que nous le nous présentons en annexe **9** utilise une variété de BASIC de marque Borland. Il se trouve en effet que les classes où nous avons expérimenté ces séries de problèmes utilisaient ce langage, par suite du choix du professeur; nous présentons donc les énoncés directement sous la forme que nous avons proposé aux élèves, de façon à faciliter la lecture du chapitre suivant, qui rapporte l'observation de groupes d'élèves travaillant sur ces séries de problèmes. L'utilisation de BASIC pour l'expression des problèmes ne relève donc nullement d'une préférence particulière pour ce langage, ni de l'indication que ce langage serait particulièrement adapté au niveau où se situe cette étude. Pour ce qui nous intéresse, tout autre langage aurait convenu, pourvu qu'il dispose du jeu réduit d'instructions, et des structures algorithmiques les plus simples (alternatives, itération à un point de sortie, en début ou en fin de boucle), et qu'il soit exécutable sur les ordinateurs à disposition des élèves. L'adaptation à un autre langage tel que PASCAL (dans une variante disposant de la structure de chaîne) LSE, et même à un tableur (**Multiplan** par exemple) ou à un langage algorithmique (**ALADIN, MEDEE**)[113] se fait sans difficulté à l'exception mineure de la deuxième série qui utilise les possibilités sonores de l'ordinateur, la mise en oeuvre de ces possibilités pouvant varier d'un langage à l'autre, et d'un type de machine à l'autre.

Ainsi l'utilisation des problèmes ne se limite en aucun cas à des classes utilisant BASIC comme langage de programmation. Il se trouve que la classe où nous avons mené la pré-expérimentation utilisait un BASIC standard, avec numéros de lignes, sans structures itératives telles que: `TANT QUE` ou `REPETER... JUSQUA`.: ces structures étaient donc réalisées à l'aide de branchements (`GOTO`) conditionnels. La classe où nous avons mené l'expérimentation définitive rapportée dans le chapitre suivant utilisait un BASIC sans numéro de ligne, avec une structure itérative de forme `REPETER... JUSQUA`, s'écrivant `DO... LOOP UNTIL....`Les énoncés présentés en annexe **9** ne comportent donc pas de numéros de ligne, et utilisent pour les itérations la forme et la syntaxe connue des élèves; dans le cas de l'utilisation d'un langage avec numérotation des lignes (BASIC standard, LSE), il serait facile de remettre les numéros de ligne nécessaires. Par contre, dans ce chapitre, l'analyse a priori de ces énoncés s'adresse au lecteur de cette thèse, et non aux élèves; c'est pourquoi nous avons employé, pour les indications de solution. un langage algorithmique inspiré de [ARSAC 83], quand la clarté de l'exposé le nécessite

Nous rappelons ci dessous le jeu restreint de fonctions, et l'identificateur de chaque fonction dans différents langages , ainsi que la forme de la déclaration de type:

[113] Pour une présentation de ces langages, voir annexe **5**.

	BASIC	PASCAL (Turbo)	LSE	**Multiplan** Version Française
constante	" ..."	' ...'	'...'	
sous-chaîne	MID$	copy	SCH	STXT
concaténation	+	concat ou +	!	+
longueur	LEN	length	LGR	NBCAR
conversion chaîne→numérique	VAL	integer	CNB	CNUM
déclaration de type	$ en fin d'identificateur	string	CHAINE	

2.4 L'articulation de l'itération et des types de données dans la progression choisie conditionne l'ordre dans lequel les énoncés sont abordés

Nous concevons ces séries de problèmes comme pouvant être insérées dans un large choix de progressions pédagogiques. Néanmoins, certaines options concernant l'articulation des acquisitions peuvent conduire à proposer aux élèves ces problèmes dans un ordre différent. Pour préciser la façon dont les modifications doivent être opérées, nous distinguons schématiquement deux types de progression pour les premières acquisitions:

- une progression du premier type consisterait dans un premier temps à travailler l'ensemble des acquisitions de base dans le domaine algorithmique (séquentialité, répétition, alternative, itération simple) sur des données numériques; dans un second temps, d'autres types seraient introduits, les chaînes de caractères et les booléens par exemple, on ferait fonctionner sur ces données les structures algorithmiques, et on poserait la question de la structuration des données en réponse à des problèmes.
- une progression du second type consisterait à présenter dès l'abord, des données de différents types , et à progresser dans l'acquisition des structures algorithmiques par la résolution de problèmes impliquant des données de type varié; on aborderait ainsi plus tôt la question de la structuration des données.

A la suite de l'étude initiale (particulièrement au chapitre 1), le choix entre ces deux types de progression nous apparaît comme une variable pertinente; mais il est clair que les observations de classe que nous avons mené sont insuffisantes pour tirer des conclusions sur la supériorité d'un type de progression sur l'autre. Les classes où nous avons mené les expérimentations de cette série de problèmes avaient suivi plutôt une progression du premier type: elles avaient, préalablement à l'étude des chaînes de caractères, travaillé l'itération sur des données numériques; c'est la raison pour laquelle les problèmes proposés ci-dessous utilisent l'itération dès la première série. Si la classe pour laquelle on se propose d'adapter ces séries de problèmes, suit plutôt une progression du second type, il faudra reprendre la succession des problèmes que nous proposons pour mettre de côté ceux qui utilisent l'itération, et les insérer dans une stratégie d'acquisition de l'itération.

2.5 La structuration des données est généralement amorcée dans les énoncés

On s'étonnera peut-être de ce que nous ne proposons pas de problèmes où les élèves auraient à mener de façon complète la construction de la structure de données et d'une représentation de cette structure dans le langage de programmation. Nous pensons, à la suite des observations qui précédent, que les élèves ne disposent pas de la maturité suffisante pour mener à bien une telle construction de façon autonome: comment envisager par exemple de représenter une file d'attente quand on ne sait pas comment retirer le premier caractère d'une chaîne, et même qu'on ne sait pas si c'est possible? On objectera que certains élèves ne rencontrent pas ces difficultés et peuvent assez vite élaborer de façon autonome une représentation du problème

assimilable par l'ordinateur. Bien qu'on ne puisse pas savoir aujourd'hui sur quels acquis antérieurs ces élèves s'appuient pour cette élaboration, les observations que nous avons menées concordent avec des observations antérieures pour montrer que ces élèves ne constituent pas la majorité du public. Il est donc clair que ces séries de problèmes s'adressent aux élèves qui ont réellement besoin d'un enseignement au niveau de l'alphabétisation, et non à ceux que leurs acquis antérieurs dispensent de cet enseignement.

Dans les problèmes que nous proposons, la structuration des données est donc largement amorcée; la représentation à l'aide de chaînes de caractères est indiquée, sauf lorsqu'il s'agit de données numériques, pour lesquelles les élèves ont à faire un choix de représentation. Nous pensons ainsi faire travailler les élèves sur des difficultés bien spécifiées, et distinguer soigneusement l'apport de l'élève de celui de l'enseignant; si, en effet, la structuration des données n'était pas amorcée, elle devrait être amenée par l'enseignant au cours de la recherche, et il est probable que dans ce cas l'enseignant apporterait également d'autres éléments de solution rendant les apports de l'élève difficiles à mettre en évidence. Notre hypothèse est que, en cherchant les problèmes que nous proposons, les élèves résolvent même partiellement les difficultés énoncées ci-dessus. Il nous semble qu'ainsi ils progressent vers la maturité qui leur permettra ultérieurement de construire de façon autonome la structuration des données dans un problème. Par ailleurs, les séries 2, 3, 4 se terminent par un problème de difficulté supérieure dont on peut penser qu'il ne sera pas résolu par tous les élèves.

2.6 Les problèmes portent sur des objets ayant une signification intuitive

La question étant celle du rapport entre objets en jeu dans un problème et données informatiques, nous nous efforcerons de construire les énoncés de façon que les objets aient du sens pour les élèves, dans un contexte intuitif:

- il pourra s'agir dans les problèmes les plus simples de l'utilisation de l'alphabet (rang d'une lettre, conversion majuscule-minuscule...), ou de jeux sur les mots (palindromes).
- dans d'autres problèmes, on proposera des situations si possible connues des élèves: utilisation des possibilités de jouer une mélodie avec l'ordinateur, numéro INSEE, file d'attente.

Il s'agit, comme nous l'avons dit plus haut, de permettre aux élèves d'évoquer des plans de ses **S.R.T.** correspondant aux situations familières, tout en imposant des différenciations de façon que l'élève progresse dans la construction de **S.R.T.** adaptés à la programmation.

3 Présentation et analyse a priori des problèmes

L'ensemble des problèmes se compose de 5 séries, conçues pour être résolues au cours d'une dizaine de séances de 1h à 1h30, et couvrant l'ensemble des types de problèmes définis au paragraphe 1. On trouvera ci-dessous l'analyse a priori de ces problèmes. Les énoncés, tels qu'ils ont été proposés aux élèves se trouvent dans le volumes Annexes (annexe 9). Il sera ainsi plus facile au lecteur de lire la suite de chapitre, et les analyses des séances de résolution au chapitre suivant, en ayant ces énoncés en regard.

3.1 Première série (ALPHA): compréhension des arguments de la fonction sous-chaîne, problème simple de "recherche translation"

Cette série concerne l'utilisation de la fonction sous-chaîne; les calculs sur les chaînes et sur les ordinaux mettent en jeu une seule variable.

ALPHA1 est du type (*CALCUL ARGUMENTS.*)
ALPHA2 est du type (*CONSTRUCTION DE FONCTIONS.*)
ALPHA3 est du type (*CONSTRUCTION ITERATION*)
ALPHA4 relève des types (*CONSTRUCTION ITERATION*) et (*CONSTRUCTION DE FONCTIONS*).

Chaque problème est présenté aux élèves sous la forme d'un programme BASIC,en deux parties:

- les lignes précédées de l'instruction REM sont comprises par le langage comme des commentaires; c'est un moyen commode de donner l'énoncé aux élèves.
- la partie programme amorce la structuration des données; elle est à compléter par l'élève.

ALPHA1: à partir de la chaîne ABCDEFGHIJKLMNOPQRSTUVWXYZ**, calculer une lettre de l'alphabet connaissant son rang** (énoncé partie Annexes page 77)

Les élèves ont ici seulement à trouver les arguments nécessaires pour isoler le caractère de rang N d'une chaîne; la chaîne est ALPHABET$, le nombre de caractères est 1; la réponse est donc:MID$(ALPHABET$,N,1)

ALPHA2: à partir de la chaîne ,
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz, **distinguer Majuscules et Minuscules** (énoncé partie Annexes page 77)

L'alternative étant construite dans l'énoncé, les élèves ont, dans la première branche de l'alternative à réutiliser la fonction d'accès à la lettre de rang N du problème précédent, avec affectation à LETTRE$. Dans la seconde branche de l'alternative ils ont à construire la fonction d'accès à la minuscule de rang N , et donc à réaliser une addition externe entre une position (l'ordinal N), et un déplacement (le cardinal 26). On attend de ce problème qu'il montre le caractère fonctionnel de MID$ (nécessité d'écrire une affectation dans chaque branche de l'alternative), et qu'il pose la question de l'ordinal, second argument de MID$, dans un cas moins trivial que le problème précédent. Une réponse possible est:

```
IF REP$="OUI" THEN LETTRE$=MID$(ALPHABET$,N,1) ELSE
LETTRE$=MID$(ALPHABET$,N+26,1)
```

ALPHA3: recherche inverse (calcul itératif du rang d'un caractère donné, dans la chaîne ABCDEFGHIJKLMNOPQRSTUVWXYZ**)** (énoncé partie Annexes page 78)

Faute de fonction permettant d'obtenir directement le rang cherché, (ce serait la fonction POSITION dont nous avons parlé plus haut), les élèves doivent mettre en évidence la nécessité d'un parcours itératif de la chaîne, et envisager le compteur comme résultat. Pour se limiter à une itération à un point de sortie, le cas d'une entrée incorrecte n'est pas envisagé. L'itération peut soit comporter toujours 26 passages dans la boucle, soit être arrêtée dès que la lettre est trouvée , ce qui donne les deux variantes suivantes:

Variante 1

```
pour i allant de  à 26
 SI sous-chaine ( alphabet$, i ,1)=lettre$ ALORS resultat← i IS
finpour
```

Variante 2

```
resultat ← 1
TANT QUE sous-chaine ( alphabet$, résultat,1)<>lettre$
 incrémenter résultat
QT
```

La deuxième variante sera présentée comme meilleure (rapidité, lisibilité). Dans les deux variantes, les élèves ont à utiliser la fonction sous-chaîne dans une expression conditionnelle; ils ont la possibilité d'employer l'affectation du résultat de la fonction à une variable intermédiaire; mais dans ce cas, le corps de boucle comporte deux instructions , ce qui pose la question de la séquentialité des instructions dans le corps de boucle, et de l'initialisation.

ALPHA4: Recherche-translation dans la chaîne
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz: **une majuscule étant donnée, calculer la minuscule** (énoncé partie Annexes page 78)

Les élèves ont à coordonner la structure itérative du problème précédent, avec la structure directe du problème **ALPHA1**; la compréhension de la signification du

compteur de boucle est également en jeu dans ce problème. Concrètement, une solution consiste à réécrire la solution de **ALPHA3**, avec la variable comme résultat, en la faisant suivre de l'affectation `RESULTAT$=MID$(ALPHABET$,N+26,1)`, déjà trouvé comme solution dans **ALPHA2**.

3.2 Seconde série (PIANO): Recherche-Translation avec arguments plus généraux

Cette série utilise la possibilité pour l'ordinateur de jouer une mélodie codée sous forme d'une chaîne de caractères pour réutiliser et généraliser les aptitudes acquises dans la série précédente.Les deux problèmes relèvent des types (*CONSTRUCTION ITERATION*) et (CONSTRUCTION DE FONCTIONS).La présentation des problèmes est la même que dans la série précédente.

PIANO1 à partir de la chaîne "`DO C RE D MI E FA F SOLE LA A SI B`", recherche de la notation anglaise d'une note de la gamme (énoncé partie Annexes page 78)

Dans la *gamme en Ut* (`DO RE MI FA SOL LA SI`) introduite au XIIIème siècle, chaque note est repérée à partir des premières syllabes des premières de l'hymne latin à St Jean-Baptiste, le Ut ayant été remplacé ultérieurement par DO. Dans la notation anglaise ou allemande, les 7 notes sont repérées par les 7 premières lettres de l'alphabet, A repérant la note LA. Le problème consiste à programmer une correspondance, ce qui se justifie par le fait que, dans le langage utilisé, l'instruction de sortie "musicale" `PLAY` prend comme argument une chaîne sous la notation anglaise. L'utilisation de la chaîne de caractères "`DO C RE D MI E FA F SOLE LA A SI B`" et d'une itération sur cette chaîne remplace l'écriture de 7 conditionnelles (qui serait la démarche spontanée de la majorité des élèves); la structure: recherche du rang du premier caractère d'une sous-chaîne donnée, calcul d'une autre sous-chaîne à partir de ce rang , est la même que dans **ALPHA4**, mais ici, la longueur de la chaîne à rechercher varie (elle vaut 3 pour la donnée SOL, et 2 pour les autres entrées), ce qui met davantage l'accent sur le troisième argument de la fonction `MID$`

La recherche du rang du premier caractère peut se faire par:

```
i←1
tant que mid$(gamme$,i,1) <> note$
  i←i+5
qt
```

et le calcul de lettre$ se complète en

```
lettre$= mid$(gamme$,i+3,1)
```

PIANO2: codage d'une mélodie à partir de l'entrée de notes par l'utilisateur (énoncé partie Annexes page 79)

Cet énoncé valorise la recherche de l'exercice précédent, en conduisant à un programme réellement utilisable; la structure sous forme de deux boucles imbriquées (une boucle principale de saisie des notes, dans laquelle s'imbrique la boucle construite au problème précédent) peut sembler difficile pour des élèves de Seconde; elle est préparée par l'écriture de la boucle principale dans l'énoncé. Du point de vue des chaînes de caractères, la difficulté nouvelle est l'utilisation d'une variable cumulative (`melodie$`) qui doit progresser par concaténation. La première ligne à compléter peut être remplacée par le module de calcul de lettre$ tiré du calcul précédent; ensuite on fait évoluer `melodie$` par:

```
melodie$=melodie$+lettre$
```

3.3 Troisième série (TESTCHA): calcul sur des ordinaux, fonction longueur, conditionnelles à plusieurs variables

Cette série se caractérise par l'utilisation de la fonction `longueur`, en particulier son intervention dans le calcul d'ordinaux. La série donne lieu également à l'écriture de conditionnelles (alternative ou itération) où interviennent plusieurs variables, et qui sont donc plus complexes que dans les séries précédentes).

TESTC1 est du type (*CALCUL ARGUMENTS.*)

TESTC2 est du type (*CONSTRUCTION DE FONCTIONS.*)

TESTC3 et **TESTC4** sont du type (*CONSTRUCTION ITERATION*) .

La présentation des problèmes est la même que pour les séries précédentes.

TESTC1: comparaison première et dernière lettre d'un mot (énoncé partie Annexes page 79)

Les élèves ont à utiliser le résultat de la fonction `longueur` pour calculer le dernier caractère de la chaîne, et à prendre en compte l'affectation à deux variables différentes pour construire une expression conditionnelle. Le programme se complète en:

```
LET A$ = MID$(MOT$,1,1)
LET B$ = MID$(MOT$,L,1)
IF A$=B$ THEN PRINT "OUI" ELSE PRINT "NON"
```

TESTC2: comparaison première, dernière et lettre du milieu (énoncé partie Annexes page 80)

Le problème nécessitant la comparaison de 3 caractères, les élèves doivent construire une affectation, et pour cela introduire une nouvelle variable de type chaîne. Ils doivent également utiliser le résultat de la fonction `longueur` dans un calcul moins trivial que le problème précédent; voici deux possibilités d'expression de la position du caractère du milieu:

- `(L + 1)/2`, expression analogue à l'expression de l'abscisse du milieu sur une droite; cette expression doit être prise dans le sens d'une moyenne entre ordinaux; il ne s'agit pas d'une addition entre ordinaux qui n'aurait pas de sens.
- `1 + (L-1)/2` où l'on reconnait l'expression de la position de l'image du dernier caractère dans l'homothétie de rapport 1/2; l'addition est ici l'addition externe entre la position du caractère de début de chaîne et un cardinal.

On complétera donc le programme par exemple en:

```
LET C$ = MID$(MOT$,(L+1)/2,1)
IF A$=B$ AND B$=C$ THEN PRINT "OUI" ELSE PRINT "NON"
```

TESTC3: comparaison (itérative) des caractères symétriques d'un mot (énoncé partie Annexes page 80)

Il s'agit de la construction d'une itération où le nombre de passage dans la boucle est connu. Le choix de donner le résultat sous forme du nombre de paires différentes, et non d'un booléen ("*Palindrome*" ou "*Non Palindrome*") résulte de la volonté exprimée plus haut de ne pas proposer de problème impliquant la construction d'une boucle à plusieurs points de sortie. La structure itérative est susceptible de prendre deux formes, que nous indiquons en langage algorithmique:

variante 1:

```
RESULTAT← 0
répéter pour I allant de 1 à L/2
 si sous-chaine (chaine$,I,1) = sous-chaîne(chaine$,L-I+1,1)
       alors incrémenter RESULTAT IS
boucler
```

Dans cette variante, l'itération se fait avec un compteur parcourant un ensemble donné; cette forme d'itération est sans doute plus familière pour certains élèves; en contre-partie la condition de l'alternative située dans le corps de boucle, utilise la construction `L-I+1` pour le calcul de la position du caractère placés symétriquement par rapport au caractère de rang `I`. Les observations précédentes laissent penser qu'une telle construction est difficile pour de nombreux élèves de Seconde.

variante 2:

```
RESULTAT ← 0
J←L
I←0
tant que I ≤ J
   si sous-chaine (chaine$,I,1) = sous-chaîne(chaine$,J,1)
         alors incrémenter RESULTAT IS
   incrémenter I
```

```
        décrémenter J
    qt
```

Dans cette variante, deux index `I` et `J` évoluent au cours du parcours de la chaîne; le corps de l'itération et la condition d'arrêt sont plus complexes; les questions de l'initialisation et de la séquentialité dans la boucle se posent dans cette variante, et l'on sait qu'elles ne sont pas simples pour les élèves. Par contre, la partie condition de l'alternative située dans le corps de boucle se trouve simplifiée.

TESTC4: comparaison (itérative) des caractères symétriques d'une phrase (sauter les espaces) (énoncé partie Annexes page 80)

La restriction "*on considérera qu'il n'y a pas plus d'un espace entre deux mots*" vise à éviter l'imbrication de boucles. Même avec cette restriction, et même si l'itération comporte un seul point de sortie, le problème est plus représentatif des difficultés de l'itération:

- le programme utilise nécessairement comme dans la variante 2 trois variables itératives (les deux index `I` et `J`, et le résultat); il faut donc articuler dans le corps de boucle les 5 actions:

```
si sous-chaîne(PHRASE$,I,1)=" " alors incrémenter I is
si sous-chaîne(PHRASE$,J,1)=" " alors décrémenter J is
si  sous-chaîne(PHRASE$,I,1) différent de  sous-
chaîne(PHRASE$,J,1) alors incrémenter RESULTAT  is
incrémenter I
décrémenter J
```

- les 5 actions ci-dessus peuvent être ordonnées de plusieurs manières pour former le corps de boucle; il est important que l'assertion *[[ni I ni J n'est la position d'un espace]]* soit vérifiée avant l'action de comparaison des caractères. La manière dont ces actions sont ordonnées est également dépendante de l'initialisation, qui est prévue dans l'énoncé.
- la construction de la condition de sortie, et son articulation avec le corps de boucle est une autre difficulté; en particulier, les conditions utilisant l'égalité (`I=J , I=J-1, I=J-2, ...`), supposent que le test de sortie soit judicieusement placé pour que la boucle termine dans tous les cas. Par contre, la condition `I > J` qui semble moins naturelle, est plus facile à placer, l'assertion *[[I inférieur ou égal à J]]* devant être vérifiée avant l'action de comparaison des caractères. Pour des élèves qui maîtriseraient bien le parcours itératif d'une chaîne à l'aide d'une boucle indexée sur un compteur (comme dans la variante 1 de **TESTC3**), on peut considérer cet problème comme une invitation à envisager l'itération de façon plus générale.

3.4 Quatrième série (INSEE): type des données, fonctions de conversion de type

Cette série est centrée sur l'utilisation de la structure chaînes de caractères pour définir des fonctions sur la représentation décimale de données numériques. La fonction de conversion `VAL` est introduite dans **INSEE2**; cette fonction fait correspondre à une chaîne le nombre dont elle constitue la représentation décimale. L'aspect (*CHOIX TYPE*) est donc dominant, mais on trouve également l'aspect (*CONSTRUCTION DE FONCTIONS*). Les énoncés sont conçus pour être communiqués aux élève sur papier, et non sous la forme de commentaires dans un programme à compléter. Nous avons jugé fastidieux pour l'élève d'entrer en tant qu'utilisateur lors des essais d'exécution à plusieurs reprises des numéros de 13 chiffres; il a semblé de même intéressant de fournir un jeu de numéros dont certains comportent des erreurs que l'élève n'aurait pas pensé à introduire; c'est pourquoi nous avons proposé aux élèves un module à placer au début de leur programme: ce programme tire au hasard un numéro à 13 chiffres parmi une liste de 6 numéros et le range dans la variable `INSEE$`; ce module s'écrit:

```
LISTE$= "1480575114227 2500514142223 2781350114225
3881276061454 1451250478962 2980361115024"
T= int(timer): Q=int(T/6): R=T-6*Q
I=14 * R + 1
INSEE$ = MID$(LISTE$,I,13)
cls
```

```
Print INSEE$
```

Il a été demandé aux élèves de le charger à partir de la disquette sans chercher à en comprendre le détail. La variable INSEE$ ayant recue une valeur parmi les 6 de la chaîne LISTE$, les élèves peuvent ensuite la traiter comme si la valeur avait été entrée par lecture.

INSEE1: contrôle de cohérence du numéro INSEE. Calcul du département de naissance (énoncé partie Annexes page 81)

Le numéro INSEE est un numéro matricule déterminé par l'Institut National de la Statistique et des Etudes Economiques pour représenter un individu [CHAMBADAL 83]. Il se présente comme la concaténation de données numériques entières écrites sous forme décimale. Les traitements visant à extraire une des données supposent de le considérer comme une chaîne. D'autres traitements, par exemple ci-dessous, le calcul de la clé de contrôle, obligent à le considérer comme une entité numérique[114]. Les élèves ont, dans ce premier problème à reconnaître la nécessité de le traiter comme une chaîne.

Bien que les résultats demandés puissent représenter des quantités numériques, on peut se contenter de fonctions d'accès retournant des chaînes; la comparaison pour la vérification du mois de l'année peut se faire de façon un peu fastidieuse en utilisant l'égalité:

```
LET MOI$=MID$(INSEE$,4,2)
LET PC$=MID$(MOI$,1,1)
IF PC$ <> "0" AND PC$< > "1" THEN PRINT "ERREUR"
IF MOI$ = "00" THEN PRINT "ERREUR"
IF PC$ ="1" AND MOI$ <>"10" AND MOI$ <>"11" AND MOI$ <>"12"
THEN PRINT "ERREUR"
```

On peut raccourcir en utilisant la comparaison des chaînes sous forme alphabétique, qui donne ici le même résultat qu'une comparaison des nombres dont les chaînes sont l'écriture décimale:

```
LET MOI$=MID$(INSEE$,4,2)
IF MOI$ > "12" OR MOI$ = "00" THEN PRINT "ERREUR"
```

INSEE 2: calcul de l'age, un numéro étant donné (utilisation d'une fonction de conversion de type) (énoncé partie Annexes page 81)

Ici, on veut faire des calculs (numériques) sur les données extraites; la conversion de type s'impose donc; le but de ce problème est de reconnaître la nécessité de cette conversion de type et de la faire manipuler dans un cas simple. Les élèves prennent connaissance de la fonction de conversion de type en répondant aux questions posées par un programme **TESTVAL**, préparé à cette intention qu'il leur suffit de charger à partir de la disquette. Nous donnons ci-dessous le listing de ce programme et un exemple d'exécution.

```
REM essai de la fonction VAL
PRINT "Entre un premier nombre"
INPUT A$
PRINT "Attention je le garde dans une variable chaîne A$"
PRINT "Entre un deuxième nombre"
INPUT B$
PRINT "Attention je le garde dans une variable chaîne B$"
  A=VAL(A$): B=VAL(B$): REM A et B sont des nombres
  X$= A$ + B$
  X= A + B
DO
```

[114] En fait, le numéro INSEE est un rassemblement de données hétérogènes (sexe, age...) dans une même structure. Il s'agit donc d'un enregistrement (Structure *RECORD* , introduite dans le langage COBOL Cf [HOROWITZ 83] p.147). Les exercices qui suivent peuvent donc être considérés comme une réalisation d'une structure d'enregistrement à l'aide de la structure "chaîne de caractères" : accès aux composants (en respectant leur type) dans INSEE1, INSEE2, construction d'une valeur dans INSEE3...

```
    PRINT:PRINT "Dis-moi ce que vaut A$ + B$"
    INPUT REP$
    IF X=VAL(REP$) THEN PRINT "Il me semble que tu confonds la
concaténation et l'addition"
    IF REP$ <> X$ THEN PRINT "Je ne comprends pas ta réponse"
LOOP UNTIL REP$=X$

PRINT:PRINT "Une ligne de mon programme s'écrit:A=VAL(A$) :
B=VAL(B$)"

DO
    PRINT:PRINT "Dis-moi ce que vaut A + B"
    INPUT REP$
    IF VAL(REP$)<>X THEN PRINT "Je crois que tu n'as pas compris la
fonction VAL"
LOOP UNTIL VAL(REP$)=X

PRINT "Bravo, terminé !".
```

Voici un exemple d'exécution de **TESTVAL**: le texte affiché par l'ordinateur est en **gras**, les réponses entrées par l'élève sont en *italiques*:

Entre un premier nombre *371*
Attention je le garde dans une variable chaîne A$

Entre un deuxième nombre *121*
Attention je le garde dans une variable chaîne B$

Dis-moi ce que vaut A$ + B$ *492*

Il me semble que tu confonds la concaténation et l'addition

Dis-moi ce que vaut A$ + B$ *371121*

Une ligne de mon programme s'écrit:A=VAL(A$): B=VAL(B$)

Dis-moi ce que vaut A + B *492*

Bravo, terminé !

Après exécution de **TESTVAL**, les élèves doivent:

- calculer la chaîne des chiffres de l'année de naissance:
  `LET ANNEE$ = MID$(INSEE$,2,2)`
- convertir en donnée numérique: `LET ANNEE = VAL(ANNEE$)`
- calculer l'age `LET AGE=1988 - ANNEE`
- corriger dans le cas où le résultat serait négatif:
  `IF AGE < 0 THEN LET AGE = AGE + 100`

INSEE 3: calcul du numéro INSEE à partir de données entrées par l'utilisateur (concaténation; variables et constantes)(énoncé partie Annexes page 82)

Ce problème est une autre occasion d'expérimenter la nécessité d'employer des variables de type chaîne, y compris pour des données d'apparence numérique si l'on veut pouvoir les concaténer; c'est d'ailleurs la première expérience d'utilisation de la concaténation pour les élèves (les moins rapides) qui n'auraient pas abordé **PIANO2**. Par ailleurs, ce problème impose de faire des calculs combinant des chaînes variables et des constantes.

INSEE4: calcul de la clé de contrôle (coordination entre la représentation du numéro INSEE comme suite de chiffres, et sa valeur numérique) (énoncé partie Annexes page 82)

Dans ce problème, le numéro INSEE est considéré comme un nombre entier pour les besoins d'une procédure formelle de vérification. Mais les contraintes du langage empêchent de considérer de façon exacte un nombre entier supérieur à 32767; les

élèves doivent donc considérer une représentation du numéro INSEE à l'aide de trois variables numériques. Ce n'est pas le procédé de calcul qu'il s'agit de trouver ici; c'est pourquoi ce procédé est indiqué dans l'énoncé (sa recherche trouverait sa place dans un problème d'un cours de mathématiques portant sur les congruences); c'est par l'expression de ce procédé sous une forme assimilable par le dispositif informatique, que les élèves manipulent la donnée sous ses différentes représentations, et c'est ce point de vue qui nous intéresse.

Concrètement, les élèves ont à construire trois fonctions d'accès, puis à utiliser successivement les fonctions VAL et MOD à bon escient:

```
LET R1$=MID$(INSEE$,1,5)
LET R2$=MID$(INSEE$,6,5)
LET R1$=MID$(INSEE$,11,3)
LET R1 = VAL(R1$)
LET R2 = VAL(R2$)
LET R3 = VAL(R3$)
LET R1 =R1 MOD 97
LET R2 =R2 MOD 97
LET R3 =R3 MOD 97
LET R= R1 * 81 + R2 * 9 + R3
LET R =R MOD 97
LET CLE = 97 - R
```

3.5 Cinquième série (FILE): utilisation de la concaténation, coordination de structures algorithmiques complexes

Cette série est constituée d'un seul problème (énoncé partie Annexes page 82); le but est de proposer aux élèves une situation où une chaîne évolue:

- par adjonction d'un caractère,
- par suppression d'un caractère.

de façon à ce qu'ils réalisent les instructions de modification adaptées,en utilisant les fonctions sous-chaîne, longueur, et concaténation. Le problème s'apparente donc au type (*CONSTRUCTION FONCTIONS*); mais ici il s'agit de construire plutôt des *instructions de modification* sur une variable d'accumulation, donc de bien distinguer les instructions des fonctions. Le problème est proposé comme pour les premières séries sous forme d'un programme avec commentaires, à compléter. On a choisi de faire évoluer la chaîne comme une file de données FIFO (first in, first out: premier entré premier sorti), car c'est certainement la plus familière aux élèves; le support intuitif d'une salle d'attente de médecin doit permettre de bien distinguer les deux phases (entrée d'une donnée, sortie d'une donnée), pouvant intervenir de façon aléatoire. Concrètement, les élèves n'ont pas en charge la construction de la structure du programme, mais seulement la rédaction des deux modules, le premier correspondant à l'entrée d'une donnée dans la file, le second à la sortie, et à les coordonner avec la structure d'ensemble du programme. Ils ont, comme indiqué plus haut, à rédiger les instructions de modification nécessaires, mais aussi à prendre en charge la gestion de la donnée élémentaire (affichage dans le cas de la sortie d'une donnée, lecture au clavier dans le cas de l'entrée).

Dans le premier module, la suppression d'un caractère posera une difficulté importante: on pourra écrire ce module (en faisant l'hypothèse que les données entrent à droite et sortent à gauche):

```
PRINT mid$(liste$,1,1)
LET liste$=mid$(liste$,2,L-1)
```

Il sera nécessaire, pour ce module, d'assurer la mise à jour de la variable L, ce qui peut se faire soit dans chaque module (incrémentation dans le second module, décrémentation dans le premier), soit en la calculant à chaque utilisation à l'aide de la fonction LONGUEUR. Pour compléter, on prendra en compte dans ce premier module le cas où il n'y a plus de patient en attente: le module s'écrira:

```
L = LEN(LISTE$)
IF L = 0 THEN
  PRINT "Plus de patient en attente"
  ELSE
  PRINT mid$(liste$,1,1)
```

```
    LET liste$=mid$(liste$,2,L-1)
  ENDIF
```

La question posée dans le second module est l'adjonction d'un caractère. On peut penser que ceci se fera facilement en utilisant la concaténation. Si l'on fait par exemple le choix d'ajouter les données par la droite de la chaîne, ce module se complète de la façon suivante:

```
PRINT "Tapez votre lettre-code"
INPUT LETTRE$
LET LISTE$=LISTE$ + LETTRE$
```

Ce second module étant le plus simple, il sera intéressant que les élèves le traitent en premier.

Chapitre 10 Expérimentation

Travail dans une classe de Seconde "option informatique"
L'évolution des difficultés et des représentations
La situation pédagogique

Nous avons proposé au cours de l'année 1988-1989, les 5 séries de problèmes présentées au chapitre précédent, dans une classe de Seconde "option informatique" du Lycée Allende d'HEROUVILLE (Calvados). Deux groupes de deux élèves ont fait l'objet d'une observation particulière; une partie des élèves de la classe a passé une épreuve d'évaluation à la suite de la résolution des séries de problèmes. Le but de cette expérimentation est de vérifier que les problèmes permettent effectivement à des élèves l'acquisition de compétences en programmation concernant les chaînes de caractères, à travers l'évolution de leurs représentations des objets et des traitements, mais aussi de développer des capacités générales en informatique.

Nous avons en effet constaté que l'évolution des représentations des élèves se fait lentement, et que des représentations non encore suffisamment différenciées des représentations naïves conduisent à un certain nombre de difficultés que nous avons repérées. Nous avons construit ces 5 séries de problèmes en fonction de ces difficultés, en faisant l'hypothèse que le fait de placer les élèves devant les difficultés, avec l'encadrement pédagogique nécessaire peut être un moyen de les faire progresser.

Les comptes rendus de 9 séances de travail des deux groupes de 2 élèves que nous avons suivis doivent servir à vérifier que les problèmes proposés placent effectivement les élèves devant les difficultés décrites plus haut, et qu'ils conduisent les élèves à une véritable activité de résolution. Nous pourrons également observer la façon dont les difficultés se hiérarchisent: certaines seront progressivement résolues au cours de l'avancement des séances, alors que d'autres persisteront chez plusieurs élèves. Nous aurons ainsi un **retour critique sur l'analyse préalable** qui a conduit à l'élaboration de ces séries.

Les comptes rendus servent également à l'analyse de la **situation d'apprentissage**: nous avons indiqué (chapitre 4) que les situations de résolution par petits groupes, encadrées par l'enseignant, l'ordinateur étant à disposition pour des essais de compilation et/ou d'exécution nous intéressent prioritairement. Nous avons, à propos de ces situations, posé la question de l'influence de **la présence du dispositif,** des contraintes d'expression et des effets en retour par lesquels il se manifeste, la question de **l'intervention de l'enseignant** auprès des groupes, et celle de la **communication** dans le groupe, et avec l'enseignant. Au chapitre précédent (§2), nous avons précisé notre choix concernant le **schéma d'apprentissage**: auprès d'élèves débutants, particulièrement ceux qui sont le plus en difficulté, l'analyse préalable à la programmation serait une exigence dénuée de sens, et donc nous préférons travailler avec eux dans le langage de programmation; les énoncés étant conçus en fonction de structures générales, et non de particularités d'implémentation dans ce langage, nous n'introduisons pas des éléments pouvant gêner pour une étape ultérieure de l'apprentissage où les éléments du langage seraient vus à un niveau plus général. Nous avons indiqué également que le recours à l'affichage de résultats partiels peut empêcher que le traitement soit véritablement considéré comme un calcul, et donc nous avons proposé l'exigence d'un traitement aboutissant à l'affectation d'une variable résultat préalable à son affichage.

Une épreuve (**EPR4**) passée par certains élèves de la classe, dont les élèves observés, a permis l'observation de productions autonomes et sans la présence de l'ordinateur à l'issue des 9 séances de résolution des séries de problèmes. Cette épreuve comporte des items déjà proposés dans d'autres épreuves rapportées en partie II (**EPR1**, **EPR2**, **EPR3**); elle sert à établir un bilan des acquisitions des élèves observés ici, et ayant travaillé sur les séries de problèmes que nous proposons, en

rapport avec les résultats obtenus en 86-87 dans une autre classe.

Les comptes rendus de séances de résolution, ainsi que le texte et les réponses des élèves à l'épreuve sont en partie annexe (annexe **10** pages 85 et suivantes). Comme nous l'avons indiqué au chapitre précédent, les textes complets des problèmes, tels qu'ils ont été proposés aux élèves, sont en annexe **9** (pages 77 et suivantes).

Ce chapitre comprend donc les parties suivantes:

1. présentation de la classe et de la progression suivie par le professeur.
2. choix des élèves dont le travail a fait l'objet d'une observation. Conditions de cette observation.
3. analyse des 9 séances de travail observées à partir des comptes rendus en annexe **10**.
4. présentation de l'épreuve sur papier.
5. compte-rendu de l'épreuve sur papier et bilan des acquisitions des élèves observés.
6. tableau récapitulatif des difficultés rencontrées par les élèves.
7. analyse par type de difficulté.
8. analyse de la situation d'apprentissage au cours des 9 séances.

1. Présentation de la classe et de la progression suivie par le professeur; contrat didactique actuel et modifications envisagées pour la résolution des séries de problèmes

1.1 Eléments de la progression

Nous donnons ci-dessous le contenu indiqué par le professeur pour les séances qui ont précède le travail sur les séries d'exercices présentées au chapitre précédent:

- éditeur du TurboBASIC (2 séances)
- survol de la structure d'un ordinateur
- programmation; analyse d'un problème(2 séances):exécution puis modification d'un programme traçant un rectangle coloré.
- l'affectation: écrire un programme qui dessine 2 rectangles prévus à l'avance, puis une figure à base de rectangles et de cercles.
- affichages (variable numérique; chaîne)
- répétition `FOR...NEXT`
- recherche: afficher une table de multiplication; faire clignoter un rectangle.
- analyse en commun du programme de déplacement d'une voiture (3 séances).
- dialoguer avec l'ordinateur (`PRINT INPUT`)
- Devoir: affichage de factures (5 séances).
- l'état des variables (faire "tourner à la main" des programmes).
- sortie sur imprimante.
- classement de 2 ou 3 nombres (2 séances).
- répéter jusqu'à; utilisation dans des programmes déjà écrits (2 séances).
- système d'exploitation (commandes de base; différents types de fichiers).
- codage en machine; écriture binaire
- écriture en binaire d'un nombre donné en base 10 (5 séances).

Nous allons brièvement analyser cette progression puis indiquer les modifications du contrat didactique que nous comptons apporter:

- la progression porte sur l'acquisition des structures algorithmiques, les données manipulées étant de type numérique: les chaînes interviennent seulement sous forme de constantes dans les sorties à l'écran.
- la répétition simple (répétition basée sur le parcours par une variable `COMPTEUR` d'un ensemble donné de valeurs) n'est pas considérée comme une structure conditionnelle; elle précède donc l'alternative (`SI... ALORS ...SINON`); l'étude de l'alternative est suivie de celle de la répétition conditionnelle sous la forme `REPETER <corps de l'itération> JUSQU'A <condition>`.
- le professeur met l'accent sur l'acquisition de l'affectation; les instructions d'affichage sont bien différenciées de l'affectation; c'est un point dont nous avons souligné l'importance au chapitre précédent; il n'y aura pas nécessité de modifier le contrat didactique à ce niveau.

- le problème (difficile à ce niveau) de la conversion décimal-binaire est motivé par la liaison avec la partie "codage en machine de l'information"; le résultat est obtenu comme variable de type numérique[115]; l'intention du professeur est de reprendre cet exercice avec un résultat sous forme de chaîne de caractères, au cours de l'étude de cette structure; les problèmes que nous proposons sont donc plus simples et plus progressifs que ceux qu'aurait envisagés le professeur.
- le professeur indique assez tôt la nécessité de **l'analyse d'un problème comme préalable à la programmation**; pour cet enseignant, l'analyse consiste en l'expression en langage naturel du procédé de calcul ou de construction qu'utiliserait l'élève en l'absence de machine; il s'agit même d'une heuristique, le processus mis en oeuvre par l'opérateur humain étant un point de départ pour expliciter un processus de résolution pour la machine. Cette heuristique va dans le sens d'une évocation de plans du **S.R.T.** intuitif (Cf chapitre **2**) comme point de départ pour la constitution d'un **S.R.T.** adapté à la tâche de programmation. Elle nous semble cependant sous-tendue par l'idée que l'explicitation (en langage naturel) d'un de ces plans est à la fois le moyen et le passage obligé pour la systématisation du procédé manuel auquel ce plan s'applique, cette systématisation étant elle même une condition nécessaire et suffisante de l'écriture d'un programme. Or, d'une part, le plan peut être présent chez le sujet, tout en étant très peu explicitable. D'autre part, l'expression conduit à la systématisation seulement si le sujet est déjà capable de penser le traitement pour un dispositif général, et s'il dispose de capacités d'expression à ce niveau. Par exemple, pour la recherche d'une information dans un texte, un sujet restant dans le **S.R.T.** intuitif sera seulement capable de donner une vague indication comme «*on regarde où se trouve l'information*», ce qui ne fait pas avancer la programmation; un autre sujet pourra avoir conscience de la nécessité d'un processus itératif (qui, pensé pour un processus manuel, constitue une systématisation), mais être incapable de trouver un moyen de penser et d'exprimer ce processus à un niveau suffisamment général pour être à la fois compatible avec un procédé manuel, et avec le dispositif: il est probable que dans ce cas, il préférera s'exprimer directement dans le langage de programmation.
  A la différence de ce professeur, nous pensons que l'adaptation des plans résulte principalement de la rencontre avec les contraintes du dispositif, qu'elles apparaissent lors de l'exécution de procédures mentales permettant d'anticiper le fonctionnement du dispositif, par des exigences de logique interne, ou par des effets en retour (erreurs de syntaxe, résultats d'exécution), sans qu'il y ait nécessairement passage à l'explicitation. L'explicitation et la systématisation introduisent en fait des exigences supérieures non motivées par le but recherché. C'est pourquoi nous demanderons aux élèves de *penser a priori un traitement pour la machine*, plutôt que de tenter d'expliciter, puis d'adapter à la machine, le traitement qu'ils mettraient en oeuvre en tant qu'opérateur humain.
- Une autre méthode utilisée dans la classe consiste à "*faire tourner à la main*" les programmes, c'est à dire noter dans un tableau l'évolution des variables au cours d'une exécution simulée d'un programme; elle constitue un moyen d'analyse a posteriori d'un algorithme. Le professeur l'emploie de façon assez systématique pour la détection des erreurs de construction des itérations; mais nous verrons qu'avec les élèves les plus faibles, cette méthode est difficile à mettre en oeuvre et requiert beaucoup d'énergie de la part de l'enseignant; pour notre part, nous conduisons plus volontiers les élèves à une analyse statique de l'itération , en vérifiant avec eux la présence et le bon ordonnancement des divers éléments de l'itération: initialisation, corps de boucle, avertisseurs d'itération, condition de sortie; ces deux méthodes doivent bien-entendu être comprises comme complémentaires, et non comme exclusives l'une de l'autre.

---

[115] Ainsi l'entrée 5 donne `101` comme résultat sous la forme d'une variable numérique, dont la valeur est en fait le nombre *cent un* et non le nombre dont l'écriture binaire est 101. Cette résolution permet d'éviter le recours à un type non numérique. Elle nous semble poser problème, puisque les variables numériques,ne sont pas considérées pour leur valeur intrinséque (indépendante d'une base) ; au contraire, leur valeur est confondue avec leur forme externe (la chaîne des chiffres en base 10).

1.2 Le langage de programmation

Le langage choisi dans l'établissement est une variété de BASIC de marque BORLAND; ce langage a les particularités suivantes: le programme est divisé en lignes, mais les lignes peuvent ne pas être numérotées. Le programmeur dispose d'un éditeur de texte pleine page, à la différence de LSE qui dispose seulement d'un éditeur de lignes, et d'autres versions de BASIC qui utilisent en général l'écran de sortie comme éditeur du programme (ce qui est source de confusions chez les débutants). Le langage inclut un compilateur, ce qui permet de distinguer les phases de *compilation* et *d'exécution*, et donc les erreurs de syntaxe (qui conduisent à une erreur de *compilation*) et les erreurs de conception (à l'exécution, le programme s'exécute, mais ne donne pas le résultat attendu, ou conduit à une erreur d'*exécution*); les erreurs de compilation et d'exécution sont signalées par un message en français.

Lors des séances de résolution, nous nous proposons d'utiliser les structures algorithmiques connues des élèves:

alternative:

```
IF <condition>
THEN
 <action1>
 ELSE
 <action2>
ENDIF
```

Cette construction permet que <action1> et <action2> soient composées de plusieurs instructions écrites chacune sur une ligne. Ainsi, bien que la notion de sous-programme ne soit pas connue des élèves, ceux-ci peuvent programmer des alternatives comportant des actions complexes sans recourir à des branchements.

itération

```
DO
 <action>
LOOP UNTIL
<condition>
```

Cette construction est la traduction de la structure algorithmique suivante: `répéter <action> jusqu'à <condition>`

Dans le cours sur la répétition, les élèves ont rencontré également la structure

```
pour i <- A jusqu'à B <action> finpour
```

Nous verrons que cette connaissance est ancienne. La connaissance plus récente de la structure `répéter <action> jusqu'à <condition>` s'y est en quelque sorte substituée, et les élèves ne se posent jamais la question d'un choix entre les deux structures.

2. Présentation des élèves dont le travail a fait l'objet d'une observation, et des conditions de cette observation

La classe et le choix des élèves observés

Par rapport à d'autres classes d'option informatique, cette classe présente les particularités suivantes:

- les séances prévues à l'emploi du temps (1h de cours, 1h30 de T.P.) sont banalisées; c'est à dire que les élèves sont en salle informatique, avec les machines à disposition pendant toutes les séances; le cours intervient en cas de nécessité, et le temps de travail par groupe des élèves domine largement les périodes d'intervention magistrale.
- la classe est peu nombreuse (une dizaine d'élèves), avec un nombre équivalent d'ordinateurs à disposition; les élèves travaillent le plus souvent à un ou deux sur une machine.

Il s'agit d'une classe de Seconde indifférenciée (premier niveau du second cycle général de l'enseignement long menant à un baccalauréat). Le niveau de ces classes est maintenant très hétérogène: l'entrée en Seconde peut être pour un élève le résultat d'une scolarité de premier cycle réussie; d'autres ont pu entrer en Seconde à la suite du manque de place dans l'enseignement professionnel, qui aurait constitué pour eux une orientation plus adaptée. Des élèves de cette classe peuvent donc connaître des difficultés scolaires importantes; mais il ne s'agit pas d'une situation d'échec généralisé qui les empêcherait de poursuivre des études secondaires. La classe de

Seconde présente ainsi une variété de niveaux scolaires parmi lesquels il nous faut faire un choix.

Les observations sur ce type de population, montrent généralement une différenciation nette selon les élèves, dans les acquisitions en informatique au cours des premiers apprentissages, mais aucune étude précise n'a mis en évidence des liens entre la capacité à atteindre le niveau des concepts de base en informatique à l'issue de ces apprentissages, et le niveau scolaire général.[116] Dans l'état actuel des recherches, on peut seulement penser que certains élèves disposent d'un minimum de maturité concernant certains concepts acquis dans d'autres disciplines, ou une forme particulière de compréhension de ces concepts, qui leur permet d'opérer un transfert utile en programmation; ainsi en est il par exemple, du concept de variable, ou de fonction.

Comme indiqué au chapitre précédent les problèmes que nous proposons n'ont de sens que pour des élèves dont les acquis antérieurs sont insuffisants pour que ces transferts s'opèrent directement, et donc pour lesquels un enseignement de base en informatique est nécessaire. Il nous a donc semblé que l'observation serait plus pertinente avec des élèves manifestant un net retard sur les autres élèves de la classe. Deux groupes de deux élèves ont été indiqués par le professeur comme particulièrement en difficulté par rapport aux apprentissages en informatique; il s'agit de **Karine-Natacha** d'une part, **Arnaud-Jérôme** d'autre-part.

Nous avons observé ces élèves au cours de séances dirigées par le professeur: ils prenaient peu d'initiatives, restaient souvent silencieux, et avaient énormément de difficulté à exprimer les actions devant être exécutées dans un programme simple, et à les coordonner. Nous n'avons pas cherché à préciser ou à mesurer a-priori par une épreuve le niveau initial de ces élèves; il nous semble que les observations des premières séances, rapportées ci-dessous, suffisent à montrer la nature et l'ampleur de ces difficultés. **Arnaud** redouble la classe de Seconde; il a déjà suivi l'option informatique l'année précédente; on verra, au cours des observations, qu'il ne montre pas de capacités qui le distingueraient des autres élèves, et qui auraient été acquises l'année précédente.

Les conditions d'observations

Les séances se sont déroulées dans le cadre de travail habituel. En tant qu'auteur de cette thèse, nous avons travaillé seul à l'observation et à l'intervention auprès des deux groupes d'élèves, le professeur de la classe étant cependant intervenue à certains moment auprès d'un groupe. Le travail des élèves et les interventions du professeur et nos interventions ont été enregistrés au magnétophone; les différentes versions des programmes ont été prises en note sur papier, ainsi que les essais d'exécution, et les messages d'erreur à la compilation ou à l'exécution. Chaque séance dure 1h (50 minutes utiles) ou 1h30 (80 minutes utiles). Nous n'avons pas établi un dépouillement intégral des enregistrements et des notes, mais un "compte rendu" rédigé immédiatement après la séance, à l'aide de l'enregistrement et des notes. Il s'agit en effet ici d'évaluer dans la durée, les acquisitions des élèves et d'examiner une situation d'apprentissage; la forme "compte rendu" permet une lecture plus facile. Nous rappelons que l'annexe **9** contient les énoncés tels qu'ils ont été proposés aux élèves, et l'annexe **10**, les comptes rendus des séances. Le paragraphe 4 de ce chapitre est constitué d'une analyse de chaque séance, à partir du compte rendu.

Les deux groupes d'élèves observés sont parmi les moins rapides de la classe; c'est pourquoi ils n'ont pas abordé les problèmes suivant: **PIANO2** (2ème série) **TESTC4** (3ème série) **INSEE4** (4ème série); on ne s'étonnera donc pas de ne pas

---

[116] Sur une population encore assez limitée, nous avons vu au chapitre 8 que les élèves manifestant dans une épreuve (EPR3) un minimum de compréhension de la structure de chaîne sont très majoritairement orientés en Première Scientifique à l'issue de la classe de Seconde, ce que nous prenons comme indice d'une scolarité de seconde réussie. Cependant, la moitié des élèves orientés en Première Scientifique ne manifestent pas ce minimum de compréhension, et l'épreuve montre des différences selon les classes qui semblent plus importantes que celles qui ont trait au niveau des élèves.

trouver mention de ces problèmes dans les comptes rendus des séances de résolution.

3. Analyse des séances de résolution

Séance 1: 8 mars 1989 durée 1h Présentation des notions par le professeur, et résolution de ALPHA1 et ALPHA2

La séance commence par une présentation des fonctions sur les chaînes par le professeur. Parmi les difficultés que nous avons rencontrées dans nos observations antérieures certaines sont soulignées par le professeur dès cette présentation:

- la notion de type (en tant qu'opérations qu'il est possible de faire sur une variable, indépendamment de l'aspect externe de cette variable).
- le caractère fonctionnel de sous-chaîne, en rapport avec les acquisitions possibles des élèves en mathématiques.

D'autres difficultés ne sont pas évoquées dans cette présentation:

- le fait que les fonctions ne modifient pas leurs arguments
- la différence de statut des deux arguments numériques (le second étant ordinal, et le troisième cardinal).

Lors de la résolution de **ALPHA2** (la variable chaîne ALPHABET$ contenant la suite ordonnée des lettres majuscules de l'alphabet suivie de la suite ordonnée des lettres minuscules de l'alphabet, rendre, au choix de l'utilisateur, la majuscule ou la minuscule de rang donné) , la première écriture de **Karine**
IF REP$="O" THEN LETTRE$= MID$(ALPHABET$,N,MAJUSCULES) ELSE
LETTRE$=MID$(ALPHABET$,N,MINUSCULE)
montre que pour cette élève, la fonction MID$ est un élément linguistique permettant de communiquer avec l'ordinateur, de la même façon qu'avec un interlocuteur intelligent et connaissant le contexte; on notera que cette conception s'impose ici, alors qu'au problème précédent, la réponse était correcte; ici l'élève ne trouve pas de façon évidente un procédé de calcul conforme à une conception correcte du dispositif; c'est alors qu'apparaît la conception erronée.

La seconde écriture `LETTRE$=MID$(ALPHABET$,N,N+26)` s'observe dans les deux groupes; elle montre une absence d'acquisition du sens du 3ème argument . Elle peut s'interpréter comme: "c'est la Nième lettre de l'alphabet, mais attention, dans la chaîne, elle se trouve à la place `N+26`"; ce troisième argument était peu critique dans le problème précédent, et sa signification peut ne pas être apparue à cette occasion. L'écriture montre donc que le second argument a été mal compris, malgré le succès au problème **ALPHA1**: ce second argument apparaît pour les élèves lié au sens qu'on attribue au caractère dans le contexte (sa place dans l'alphabet) et non comme une donnée intervenant dans un calcul: une majuscule et la minuscule correspondante se voient attribuer la même place, même si leur rang diffère dans la chaîne. Le troisième argument se voit confier le rôle d'indiquer le genre (majuscule ou minuscule) du caractère. On remarquera chez **Karine** que la formulation employée pour ce troisième argument évolue: c'est d'abord un identificateur qui a un sens pour l'élève, mais pas pour le dispositif, puis le résultat d'un calcul assimilable par le dispositif. Mais la signification qu'elle attribue aux arguments numériques de la fonction sous-chaîne n'a pas évolué: le second argument indique la position dans l'alphabet, et non la position dans la chaîne, et le troisième est une correction apportée pour tenir compte du genre (majuscule ou minuscule) du caractère.

Les difficultés rencontrées dans **ALPHA2** apparaissent cependant comme temporaires: dans le cas de **Karine**, le programme est corrigé après une hésitation; **Jérôme** et **Arnaud** corrigent leur programme après une exécution qui leur permet de mettre en évidence la signification du 3ème argument.

Séance 2:13 mars Durée: 1h30 Résolution de ALPHA3 puis de ALPHA4

ALPHA3: il s'agit de trouver le rang d'une lettre dans l'alphabet, en parcourant une chaîne `ALPHABET$` ayant comme valeur "`ABCDEFGHIJKLMNOPQRSTUVWXYZ`".
ALPHA4: `ALPHABET$` a cette fois comme valeur une chaîne formée des 26 lettres de l'alphabet majuscules, suivies des 26 lettres de l'alphabet minuscules; on demande

d'entrer une majuscule, et d'afficher la minuscule correspondante.

Au cours de cette séance, apparaissent des difficultés concernant l'affectation:

- affectation sans signe "=": `LET X+1`
- affectation qui ne prend pas en compte les contraintes du dispositif informatique: `LET LETTRE$=X` qui à pour sens "X est le rang de `LETTRE$`"
- confusion entre affectation et condition d'arrêt.

Le groupe **Jérôme-Arnaud** rencontre, avec l'affectation à `LETTRE1$` de la Xième lettre de `ALPHABET$` deux difficultés supplémentaires: la séquentialité des instructions dans le corps de boucle, et l'initialisation.

Les tentatives (non-concluantes) de résolution de **ALPHA4** montrent une difficulté surprenante à séparer la question en deux sous-problèmes chacun d'eux ayant été résolu dans les problèmes précédents. **Karine** se montre incapable de distinguer ces deux phases, bien qu'elle soit très guidée dans cette direction; sa dernière écriture: `LOOP UNTIL LETTRE$= MID$(ALPHABET$,X+26,1)` révèle une confusion totale entre la condition d'arrêt (qui donne à `X` son sens, comme résultat de l'itération), et le calcul du résultat (qui utilise `X`), et l'impossibilité de les séparer. On peut penser que l'élève, même si elle construit une boucle suite à nos indications, ne comprend pas réellement cette construction; en particulier, l'expression qui suit `UNTIL` n'est pas comprise comme une condition de sortie. L'élève concentre son attention sur l'expression `MID$(ALPHABET$,X + 26,1)` qui est pour elle l'expression du résultat; le fait que cette expression soit incluse dans une affectation, une condition, ou une instruction d'affichage, échappe à son analyse. L'autre groupe rencontre les mêmes difficultés à réutiliser les résultats des problèmes précédents.

Nous nous étions donné comme tâche d'observer comment des messages d'erreur du compilateur exprimés en Français peuvent être une aide aux élèves: nous constatons que le diagnostique d'erreur (*"opérateur de comparaison attendu dans expression booléenne"*) donné par le compilateur pour l'expression: `LOOP UNTIL X= MID$(ALPHABET$,X+26,1)` n'est pas interprétable par l'élève. Par ailleurs il s'agit d'une indication d'ordre syntaxique, alors que l'erreur de l'élève est en rapport avec la sémantique de l'affectation. L'erreur d'exécution "*appel de fonction invalide*", qui survient ensuite, a un rapport plus direct avec l'erreur d'initialisation dont elle est la conséquence; mais cette erreur se trouve éloignée physiquement de l'instruction pointée par le message.

Séance 3: 20 mars 1989; durée 1h15 Résolution de PIANO1

Il s'agit, à l'aide de la chaîne `GAMME$` ayant pour valeur `"DO C RE D MI F FA G SOLH LA A SI B FIN "` d'entrer une chaîne de deux ou trois caractères (`DO, RE ... SI`), de calculer la lettre correspondante (`C ,D , ..., A, B`) et de la faire jouer par l'ordinateur.

On retrouve les difficultés rencontrées pour la résolution de **ALPHA3**: le passage par le calcul d'un rang pour construire une fonction ayant comme argument une chaîne et rendant une chaîne; on rencontre chez **Jérôme-Arnaud** la confusion entre condition de sortie de la boucle et calcul du résultat observé chez **Karine** dans **ALPHA4**; par contre les difficultés spécifiques au problème (recherche d'une occurrence d'une chaîne de longueur variable, fonction longueur) apparaissent secondaires. L'écriture du troisième argument dans `MID$` continue également à être une difficulté, mais plus passagère.

Concernant l'itération, on remarquera que les élèves utilisent la forme `REPETER <action> JUSQU'A <condition>` et non une forme plus lourde constituée d'une itération sur tous les caractères de la chaîne; la forme `REPETER <action> JUSQU'A <condition>` met l'accent sur `<condition>` comme assertion vérifiée en sortie de boucle; les élèves ont tendance à inscrire comme `<condition>` ce qui concerne le résultat final demandé dans l'énoncé; cette heuristique fonctionne parfaitement bien dans **ALPHA2** , où c'est précisément le compteur de boucle qui est le résultat, mais échoue dans le cas où il constitue un résultat intermédiaire.

Les élèves sont en fait très peu capables d'imaginer comment fonctionne une telle boucle (valeurs successives prises par les expressions intervenant dans la condition). Le type du compteur, c'est à dire de la position de la sous-chaîne recherchée, n'est pas

clair pour les élèves: pour l'ordinateur, c'est un nombre; mais pour le programmeur, c'est un ordinal, et pour les élèves il y a confusion entre cet ordinal et le caractère qu'il pointe; ce peut donc être une chaîne, ainsi que l'indique l'écriture de **Karine-Natacha**: `UNTIL RANG$=MID$(GAMME$,X,1)` Nous faisons l'hypothèse que, comme nous l'avons vu chez d'autres élèves en partie II, cette incertitude conduit les élèves à considérer que cet ordinal peut s'obtenir "directement", c'est-à-dire sans itération, et intervient donc dans les difficultés qui suivent l'indication pressante donnée par l'enseignant de construire une itération.

Séance 4; 22 mars 1989; durée 1h Résolution de TESTC1 et TESTC2

TESTC1: il s'agit d'écrire un programme testant si la première et la dernière lettre d'une variable chaîne d'identificateur `MOT$` sont les mêmes. La variable `L`, résultat de la fonction longueur est correctement employée dans **TESTC1** , bien qu'on trouve chez **Karine** des hésitations qui rappellent celles rencontrées au chapitre 6: le dernier caractère est-il de rang `L` ou `L-1` ?

TESTC2: il s'agit ici, dans le cas d'un mot de longueur impaire, de tester si la première lettre, la dernière lettre et celle du milieu sont les mêmes. La confusion entre "moitié" est "milieu" apparaît comme une forme de la confusion entre ordinal et caractère; une contradiction vient de ce que l'ordinal doit être entier; l'écriture est donc corrigée, mais sans qu'apparaisse l'expression `(L + 1)/2`, que les élèves pourraient déduire d'acquisition en géométrie (abscisse du milieu d'un segment dont les extrémités ont pour abscisse `1` et `L`). Le test sur 3 lettres avait été introduit pour que les élèves soient conduits à construire une affectation de façon complète, y compris en introduisant un identificateur de variable chaîne non prévu dans l'énoncé: ceci est fait sans difficulté. Par contre les élèves rencontrent une difficulté dans l'écriture d'une condition portant sur trois variables; mais ils résolvent cette difficulté de façon autonome.

Séance 5: 17 avril; durée 1h30 Résolution de TESTC3

(Il s'agit de vérifier la qualité de "palindrome" d'une chaîne, en comptant le nombre de paires symétriques différentes.) Malgré l'énoncé, les élèves tentent de parcourir la chaîne à partir du milieu; le second groupe y parviendra, mais au prix d'une restriction de la généralité du programme: le programme fonctionne seulement dans le cas d'une chaîne de longueur paire. Il faut supposer que pour ces élèves, le mot "symétrique" renvoie à "centre de symétrie" qu'ils savent trouver depuis le problème précédent, et que ce mot constitue pour eux une indication plus importante que le parcours amorcé dans l'énoncé. De même, les élèves n'utilisent pas la possibilité de calculer le rang du caractère symétrique d'un caractère de rang donné; ils utilisent l'itération à deux index évoluant parallèlement, dans une forme proche de la variante 2, que nous avons indiquée au chapitre précédent dans l'analyse a priori de **TESTC3**. Il est probable que pour ces élèves, le calcul du rang du caractère symétrique d'un caractère de rang donné est difficile, et peut-être même ne savent-t'ils pas que ce calcul est possible. La forme d'itération qu'ils utilisent, multiplie les difficultés concernant les différents éléments de l'itération, ce qui donne lieu à des écritures incorrectes où transparaissent des conceptions erronées du dispositif; ces éléments sont corrigés l'un après l'autre, mais le plus souvent à l'aide d'une intervention d'un enseignant.

Séance 6: 12 avril 1989; durée 1h Résolution de INSEE1

(Il s'agit de vérifier que le premier chiffre d'une chaîne composée de 13 chiffres est 1 ou 2 et que les 3ème et 4ième chiffres forment le numéro d'un mois de l'année, et d'afficher le numéro du département de naissance). La fonction `MID$` est spontanément utilisée par les élèves pour isoler une partie du numéro; les erreurs de type qui interviennent dans les premiers programmes écrits par les élèves diffèrent d'un groupe à l'autre: erreur d'expression des constantes dans le cas de **Karine-Natacha**, erreur de déclaration de type des variables (pas de signe "$" à la fin des identificateurs) pour **Jérôme**. Ces erreurs de type sont sanctionnées par un message d'erreur du compilateur; ce message est plus explicite dans le cas de l'erreur commise par

Karine-Natacha que dans le cas de **Jérôme**.

Même après avoir écrit correctement les identificateurs de variable et les constantes, les élèves appliquent aux chaînes des opérateurs d'inégalité qui relèvent du numérique (alors qu'une comparaison à l'aide de l'égalité caractère par caractère aurait été possible, bien que plus lourde). Il se trouve que cette mauvaise conception de départ conduit cependant à un programme qui fonctionne. L'opérateur d'inégalité effectue en effet dans les chaînes une comparaison selon l'ordre alphanumérique, compte-tenu d'un alphabet où les chiffres seraient rangés avant les lettres, dans l'ordre croissant; ainsi, si une chaîne de chiffres `A$` est avant une chaîne de chiffres `B$` ayant la même longueur, le nombre dont `A$` est l'écriture décimale est plus petit que le nombre dont `B$` est l'écriture décimale. C'est pourquoi le professeur et moi-même tentons de mettre en cause la conception des élèves avec des exemples de comparaison de chaînes de caractères. Des exemples, ou un problème dans les chaînes de chiffres pourraient plus utilement intervenir sur cette conception; ainsi la proposition `"123" < "22"` est vraie.

Séance 7: 17 avril; durée:1h30 Résolution de INSEE2 et INSEE3

(Dans **INSEE2**, les élèves doivent calculer l'age au premier janvier 89 du titulaire d'un numéro INSEE; ils prennent connaissance de la fonction de conversion de type `VAL`, en exécutant un programme appelé **TESTVAL**; dans **INSEE3**, ils ont à former le numéro INSEE d'une personne après avoir entré les renseignements le concernant.) Des erreurs d'écriture de l'affectation apparaissent à nouveau chez **Karine-Natacha**, à l'occasion de l'emploi de la fonction `VAL: 89-D=VAL(D$)` et d'une conditionnelle: `IF D$ > "89" THEN 100 + E`. La première écriture signifie nettement: on veut que la différence de `89` et de `D` soit `VAL(D$)`; il s'agit d'une généralisation qui pourrait se concevoir[117]. La seconde erreur est l'oubli déjà observé chez ces élèves de la première partie de l'affectation quand la variable résultat est un opérande de l'expression.

Les difficultés à écrire les constantes chaînes se retrouvent également ici, pour ces élèves, mais elles se traduisent par l'adjonction d'un signe "$", par confusion avec les identificateurs de variables; on remarquera que le compilateur signale une erreur de syntaxe pour `1$`, ce qui conduit les élèves à corriger de façon autonome, mais il n'y a pas de message pour `M$` car le BASIC ne connaît pas l'erreur: "*variable non déclarée*", ce qui rend cette erreur plus difficile à déceler. **Arnaud** et **Jérôme** ne connaissent pas ces difficultés avec l'affectation et l'écriture des constantes. Par contre, ils passent énormément de temps à chercher le moyen de corriger la valeur obtenue pour l'age dans le cas d'une année de naissance supérieure à 89, qui correspond à une naissance au XIXème siècle; c'est une autre illustration du manque de familiarité d'élèves de Seconde avec les ordinaux; il faut en effet revenir avec eux aux dates absolues (18xx , 1989) pour qu'ils aperçoivent le calcul à faire.

Séance 8: 19 avril 1989; durée: 1h Résolution de FILE

Jérôme-Arnaud: Suite de la résolution de **INSEE3**: **Jérôme** et **Arnaud** rencontrent ici des difficultés à distinguer les constantes et les identificateurs de variables; cette difficulté est liée à la non-disponibilité de la concaténation, et peut-être aussi à l'instruction d'affichage `PRINT` (les élèves ont sans doute rencontré des expressions telle que `PRINT "Le résultat est"; X` , où constantes chaînes et identificateurs de variables numériques sont affichés dans la même instruction); cette difficulté disparaît avec l'indication de calculer le résultat à l'aide de la concaténation, avant de l'afficher.[118]

---

[117] En langage déclaratif tel que PROLOG, cette écriture donnerait effectivement une valeur à `D`, mais une telle généralisation demanderait beaucoup de précautions : existence d'une opération inverse et d'un algorithme pour cette opération, indication précise de la variable à laquelle donner une valeur; nous examinerons ce point dans les conclusions concernant l'acquisition de l'affectation (8.2.b).

[118] L'utilisation d'instructions d'affichage sans passage à la ligne, ou l'affichage de plusieurs

Karine-Natacha Résolution de **FILE** (il s'agit de compléter un programme gérant la file d'attente que constitue la salle d'attente d'un médecin; les élèves ont à compléter deux modules: l'un, correspondant à une action du médecin, doit afficher la lettre-code du patient le plus ancien dans la salle d'attente, et mettre à jour la file; l'autre correspond à l'action d'un patient arrivant dans la salle: il entre sa lettre-code, et l'ordinateur met à jour la file.) **Karine** et **Natacha** complètent sans grande difficulté le second module après qu'on leur ait montré que la concaténation est l'opération adéquate. Par contre elles rencontrent une difficulté significative avec le premier module: l'action du médecin indiquée dans l'énoncé (il appuie sur la touche "-"), comme l'intuition les conduit à tenter l'utilisation de l'opérateur "-"; mais elles reconnaissent que cet opérateur n'a pas de sens ici; par contre la fonction sous-chaîne (`MID$` en BASIC) n'est pas suffisamment disponible pour qu'elles puissent l'utiliser pour calculer la partie de la chaîne à partir du second caractère; on remarquera que les élèves se centrent sur "le premier caractère", et n'envisagent pas la file mise à jour comme calculée à partir de la file actuelle; l'expression `MID$(A$, L-1,1)` signifie sans doute qu'on retire un caractère (`L - 1`) et que ce caractère est la valeur de `A$`.

Séance 9: 24 avril 89; durée 1h30mn. Résolution de FILE

Fin de la résolution de **FILE**. **Natacha** passe une partie de la séance à résoudre la difficulté rencontrée la fois précédente: calculer la partie de la chaîne à partir du second caractère, ou plutôt, pour elle, "enlever le premier caractère"; ce qu'elle traduit cette fois par une écriture comportant seulement des chaînes comme opérandes: `LISTE$=LISTE$ - A$`, qu'elle rejette ensuite; elle rejette également l'expression `L-1`, ce qui indique que pour elle, `L` et `1` ne désignent pas, au premier abord, des nombres. La première écriture qu'elle produit à l'aide de la fonctions sous-chaîne correspond à la partie de la chaîne de longueur `L-1`, mais commençant au PREMIER caractère; ce qui s'explique par la centration sur le premier caractère qu'il s'agit d'enlever. La prise en compte du cas où une place est disponible, et la salle d'attente vide donne lieu aux difficultés déjà rencontrées de coordination des instructions dans les alternatives imbriquées.

Jérôme et **Arnaud** rencontrent les mêmes difficultés, mais la concaténation, et la fonction sous-chaîne sont nettement mieux acquises par **Jérôme**, ce qui fait que des écritures correctes sont obtenues assez rapidement; elles ne sont cependant vraisemblablement pas comprises de **Arnaud** .

4. L'épreuve sur papier à l'issue des séries de problèmes

Nous avons indiqué en début de chapitre les objectifs de cette épreuve, et les conditions de passation. Nous présentons ici l'énoncé.

Nous avons repris, à une modification près, des textes d'exercices proposés dans les épreuves sur papier passées dans une classe d'option informatique au cours de l'année 86-87 (épreuve 1 et épreuve 2, voir chapitre ...), pensant ainsi pouvoir marquer une différence dans les acquisitions d'élèves ayant suivi la progression d'exercices que nous proposons. L'épreuve a été passée le 26 avril 89; les élèves disposaient d'une heure. Ils pouvaient utiliser leurs notes de cours, comportant en particulier les programmes réalisés lors des séances de résolution des séries d'exercices.

Nous présentons successivement les trois questions, puis nous analysons les réponses des élèves. Nous avons répété en annexe 10 le texte de cette épreuve, de façon à faciliter la lecture de la suite de ce chapitre.

---

valeurs dans la même instruction est une facilité couramment rencontrée dans les premières pratiques de la programmation pour la sortie d'un résultat composé de plusieurs valeurs. Nous avons indiqué au chapitre précédent pourquoi nous elle nous paraît inadéquate si elle se substitue au calcul d'une chaîne de caractère à l'aide de la concaténation. Il se vérifie ici que le calcul du résultat à l'aide de la concaténation est plus opératoire pour les élèves que l'affichage de plusieurs valeurs dans la même instruction.

question 1 (**ONJOURB**)

Ecris un programme pour que l'utilisateur, ayant entré une chaîne au clavier, l'ordinateur affiche la chaîne en passant la première lettre à la fin.

Exemple: si l'utilisateur entre "BONJOUR", l'ordinateur affichera "ONJOURB". Complète le programme:

```
PRINT "Entrez votre chaîne"
INPUT CHAINE$
```

La question 1 est reprise de EPR1 (chapitre **6**); ce problème nous avait permis de mettre en évidence les difficultés liées d'une part à la compréhension des fonctions (difficultés relatives à la logique interne à la structure: *D.Sem.Cha.*), d'autre part, aux rapports entre la structure et les autres éléments du programme (notation fonctionnelle, affichage, affectation: *D.Langage*). Dans cette question, se pose également la difficulté d'obtenir la partie de la chaîne commençant à la deuxième lettre, donc à calculer les deux argument numériques de la fonction sous-chaîne; nous avons appelé cette difficulté *D.Ord.Card.*

question 2 (**SOMME**)

Ecris un programme pour que, l'utilisateur ayant entré un nombre entier au clavier, l'ordinateur affiche la somme du premier et du dernier chiffre.

Exemple: nombre 153 réponse 4

Dans l'épreuve EPR1, nous avions posé l'exercice suivant: "Ecris un programme pour que, l'utilisateur ayant entré un nombre au clavier, l'ordinateur affiche le chiffre des unités". Nous avions constaté que les 3 élèves ayant répondu à la question avaient employé la fonction "sous-chaîne" de la structure "chaînes de caractères" tout en déclarant les variables en jeu comme de type numérique; nous avions fait l'hypothèse que, plus qu'une erreur de syntaxe ou un oubli, ce comportement est révélateur d'une mauvaise conception des données; cette conception consiste à faire dépendre le choix d'un type pour représenter un objet de l'aspect externe de cet objet, et non de la façon dont il doit être représenté dans le dispositif pour qu'opèrent sur lui les fonctions nécessitées par le problème. Nous avions appelé (*D.TYPE*) cette difficulté, et construit en fonction de celle-ci la série d'exercices **INSEE** (type *CHOIX TYPE*). Cet exercice nécessitant de déclarer comme de type chaîne des variables représentatives de quantités numérique, mais aussi d'utiliser une fonction de conversion pour ensuite obtenir le résultat d'une opération numérique, permet donc une évaluation des acquis dans ce domaine.

question 3 (**CRYPTAGE**)

Deux correspondants souhaitent échanger des messages confidentiels.

C'est pourquoi ils conviennent du cryptage suivant de leurs message:

- ils conviennent d'une chaîne de caractères comportant les 26 lettres de l'alphabet dans un ordre donné; on prendra ici la chaîne "azertyuiopqsdfghjklmwxcvbna"
- le cryptage consiste à remplacer chaque lettre du message par la lettre qui la suit dans la chaîne.

Ainsi, le message "bonjour" est crypté en "npakpit".

On veut un programme tel que:

- en entrée on donne une lettre de l'alphabet,
- en sortie on obtient la lettre correspondante dans le cryptage.

Complète le programme:

```
LET ALPHABET$ = "azertyuiopqsdfghjklmwxcvbna"
PRINT "Entrez la lettre (minuscule) "
INPUT LETTRE$
.....
.....
PRINT "Voici la lettre codée"
PRINT LETTRE1$
```

Ce problème reprend la situation de la seconde question de EPR2 (chapitre **7**) qui visait à mettre en évidence les difficultés spécifiques que rencontrent les élèves lors

d'itérations simples, les objets en jeu étant des chaînes de caractères. Ce problème est proche de l'énoncé **ALPHA3** de la progression que nous proposons (il s'agit dans **ALPHA3**, de produire la minuscule d'une lettre donnée en majuscule, à l'aide d'une chaîne de caractères comprenant l'alphabet en majuscule, suivi de l'alphabet en minuscules). Les élèves doivent dans cette question mettre en évidence la recherche de la position de la lettre donnée dans `ALPHABET$` comme un élément de la solution et construire une itération. Cette itération peut prendre les formes suivantes:

- `REPETE...JUSQU'A`, la sortie de boucle se produisant lorsque le compteur d'itération repère la position de la lettre cherchée; dans ce cas, il devront, après la boucle, calculer la lettre résultat, en fonction de la position trouvée: `RESULTAT$= sous-chaîne(ALPHABET$, position + 1,1)`
- Un parcours total de la chaîne `ALPHABET$` (nombre de passages dans la boucle égal à 26) comprenant comme corps de boucle l' instruction conditionnelle:
  `SI sous-chaîne(ALPHABET$, compteur,1)= LETTRE$ ALORS RESULTAT$= sous-chaîne (ALPHABET$, compteur + 1,1)`

Bien que plus lourde, et moins efficace, la deuxième forme était largement employée par les élèves ayant passé EPR1; ils utilisaient un BASIC ne permettant pas d'autre forme structurée de l'itération. Les élèves qui passent l'épreuve ici, après un travail sur la progression d'exercice que nous proposons, disposent en principe des deux formes d'itération; nous avons constaté, au cours de la résolution de **ALPHA2** et des exercices suivants qu'ils utilisent exclusivement la forme `REPETE...JUSQU'A` (qui s'écrit dans la variété de BASIC qu'ils utilisent DO... LOOP UNTIL...), y compris pour l'énoncé **TESTCHA3** où le nombre de passage dans la boucle est connu, et où la forme `REPETE...JUSQU'A` n'apporte pas d'efficacité supplémentaire, et oblige à utiliser un index supplémentaire. Nous avions observé cependant lors de la résolution de **ALPHA3**, que la forme `REPETE...JUSQU'A` oblige les élèves à séparer la partie itérative (calcul de la position) du calcul non-itératif du résultat, et que cette séparation a été très difficile à obtenir, les élèves mêlant la condition de sortie de boucle, et le calcul du résultat.

Nous avions noté dans la présentation de EPR2 (chapitre 7),que l'exercice présente les difficultés suivantes:

- la difficulté d'envisager une itération: de nombreux élèves n'appréhendent pas la recherche de la lettre à coder dans la chaîne comme itérative. En effet, leur propre expérience, comme opérateur humain, de la recherche d'un élément dans une chaîne les conduit plutôt à l'idée d'un accès "direct"; ils produisent donc des écritures associant directement le résultat à la lettre à coder.
- la difficulté d'envisager la position du caractère courant comme de type numérique, ce qui se traduit par des écritures relevant de l'erreur de type (position = caractère, ou opération entre position et caractère); en effet la recherche par un opérateur humain d'informations dans une chaîne (ou plus généralement, une liste séquentielle) utilise la disposition spatiale des éléments, et non un compteur de position; la nature exacte de cette position échappe donc aux élèves d'autant plus que les commentaires de l'enseignant font souvent plus référence à la disposition spatiale ("avant" , "après") qu'à la valeur numérique de la position, et que le type de cette position est en fait ordinal, ce qui correspond à une variété numérique avec laquelle les élèves ne sont pas familiers.
- les difficultés générales concernant l'itération: construction de la condition d'arrêt, initialisation, séquentialité dans le corps de boucle; ces difficultés se compliquent par le fait que les constructions mettent en jeu les chaînes de caractères et les ordinaux, que les élèves ne maîtrisent pas encore.

La forme que prenait cet exercice dans EPR2 mettait l'accent sur le cas particulier que représente la donnée de la dernière lettre de la chaîne (la chaîne ne comportait que les 26 lettres de l'alphabet, et non comme ici, un 27ème caractère, égal au premier); ainsi la structure alternative de l'énoncé pouvait contribuer à masquer encore le caractère itératif de la construction demandée, et la solution complète imposait de coordonner une structure alternative et une structure itérative, ce qui n'avait été réalisé par aucun élève. Ici, la chaîne proposée dans l'amorce de programme comprend un 27ème caractère, égal au premier, de façon qu'il n'y ait pas lieu de traiter un cas particulier. Ainsi le caractère itératif de la solution à construire n'est ni

spécialement induit par l'énoncé, ni spécialement masqué, et il n'y a pas à coordonner deux structures algorithmiques.

5. Analyse des réponses des élèves à l'épreuve sur papier

Cinq élèves de la classe ont, à l'issue des 9 séances de résolution, passé l'épreuve sur papier; parmi ces élèves, trois font partie des sujets observés[119]. Le texte des réponses des élèves est en annexe **10**. Nous donnons dans ce paragraphe une analyse par élève suivie, pour les trois élèves que nous avons observé, d'un bilan comparé aux résultats obtenus dans d'autres classes.

Julien

Les réponses sont correctes; la copie a été rendue au bout de 20 minutes; **Julien** ne fait pas partie des 4 élèves observés; il domine largement la classe; on observera en particulier l'aisance que cet élève montre dans le traitement des écritures fonctionnelles; cette aisance lui permet d'éviter le recours à l'affectation.

Michaël

Michaël ne fait pas partie des 4 élèves observés (sauf à la séance 5, où **Arnaud** étant absent, il travaille avec **Jérôme**); il connait cependant lui aussi d'importantes difficultés. La réponse à la première question (**ONJOURB**) est syntaxiquement correcte; le calcul des deux éléments du résultat est correct, mais le résultat n'est pas calculé; la réponse pourrait résulter d'une confusion entre l'affichage (première ligne) et l'affectation (dernière ligne).

La seconde question (**SOMME**) est correcte.

Dans la troisième question (**CRYPTAGE**), l'élève se donne pour tâche de rechercher itérativement `LETTRE1$` (la lettre codée) sans chercher au préalable le rang de `LETTRE$` (la donnée). Il imagine pour cette recherche, de considérer successivement chaque caractère de la chaîne et de comparer le CARACTERE PRECEDENT à `LETTRE$` (la donnée). Nous avions noté lors de la résolution de **ALPHA3**, les difficultés éprouvées par les élèves pour séparer le calcul (itératif) du rang de la lettre donnée, de celui (direct) de la lettre résultat. La recherche d'un caractère dans une chaîne ou plus généralement d'un item dans une liste par un opérateur humain n'utilise pas en effet explicitement un compteur de position même dans le cas où elle se fait de façon clairement séquentielle; en effet la disposition spatiale des caractères dans la chaîne , ou des items dans la liste suffit pour la mise en oeuvre d'un procédé de recherche. L'analyse (non formulée) qui sous-tend la réponse de **Michaël** est correcte, et plus proche d'une résolution par un opérateur humain (puisque le parcours met en jeu les lettres, plus que leur position). Mais il manque à l'élève une fonction: celle qui permettrait de passer d'une lettre à la précédente; utilisant un calcul sur la position, la construction de cette fonction réintroduit les difficultés concernant les ordinaux; l'élève résout en employant une écriture n'ayant pas de sens dans le langage: `MID$(LETTRE1$,L,-1)`, mais qui, pour lui, signifie: la lettre précédant `LETTRE1$`; on reconnait comme dans d'autres observations, que, dans cette expression, le couple

---

[119] Les conditions institutionnelles de l'option expliquent ce faible effectif, déjà constaté lors d'épreuves sur papier dans d'autres classes : les effectifs de la classe peuvent être importants en début d'année, mais ils se stabilisent parfois à une dizaine d'élèves ; l'horaire de la classe est établi après celui des autres disciplines, la classe d'option regroupant des élèves de plusieurs divisions ; les séances peuvent être en fin d'après-midi, ou pendant la coupure le midi, ou le mercredi après-midi, et viennent donc en concurrence avec d'autres activités, légitimes pour les élèves : rencontres sportives, visites chez le médecin... ; ceci peut expliquer la faible participation aux épreuves sur papier ; l'appréhension d'une évaluation dans une discipline où les élèves débutent, et où l'on pratique généralement l'évaluation "continue" sans épreuve formelle peut également être une explication. Nous avons expliqué au chapitre 4 que ce type de situation s'est trouvé compatible avec le caractère "clinique" qui apparaît comme une nécessité compte-tenu de l'état actuel de la recherche. Ici, il est important pour nous que trois élèves, sur les quatre ayant fait partie de l'échantillon observé aient été présents à l'épreuve.

deuxième et troisième arguments est formé par l'élève dans le but de définir la sous-chaîne, le rôle spécifique de chacun de ces arguments n'étant pas respecté. De plus, ici le premier argument est également erroné; la syntaxe exige comme premier argument la chaîne dont la sous-chaîne est extraite; l'élève place, quant à lui, un caractère avec lequel il veut établir un rapport. Ainsi, les trois arguments sont compris comme un ensemble d'indications données au dispositif, sans prise en compte de leur rôle spécifique.

Dans la réponse apparaissent aussi des erreurs liées à l'itération: initialisation dans le corps de boucle, absence d'évolution du compteur (L), confusion entre condition et affectation: `LET LETTRE$ = MID$(LETTRE1$,L,-1)`, et donc rédaction erronée de la condition d'arrêt.

Natacha

La première question (**ONJOURB**) est correcte, si ce n'est que la variable `L` n'a pas reçu de valeur (elle désigne manifestement pour l'élève la longueur de la chaîne), et qu'il n'y a pas d'affichage du résultat. L'élève s'est appuyée sur la résolution du problème **FILE** (gestion d'une file d'attente: dernière série) pour le calcul de la partie de la chaîne au delà du second caractère, et l'ajout d'un caractère à la fin d'une chaîne; en effet on retrouve les affectations qui modifient la variable `CHAINE$`; ce type d'affectation était nécessaire dans **FILE**, alors qu'ici, l'affectation à de nouvelles variables est possible.

La réponse à la seconde question (**SOMME**) serait correcte si l'on supprimait les instructions d'entrée au clavier: `INPUT A$` et `INPUT B$` et si la variable `L` recevait par affectation le nombre de caractères de `CHAINE$` comme valeur.

La réponse à la troisième question (**CRYPTAGE**) ne comprend pas d'itération; elle repose (de façon erronée) sur une correspondance directe entre la lettre donnée et la lettre résultat; nous avions déjà repéré des réponses analogues dans EPR2; le second argument de la fonction `MID$` est une addition (sans signification dans le langage) entre une lettre et le nombre 1; il y a confusion entre un caractère et sa position dans la chaîne.

Natacha est une des élèves observées au cours de la résolution des séries de problèmes que nous avons proposé; on ne retrouve pas dans ses réponses les erreurs de syntaxe concernant l'affectation rencontrées à de nombreuses reprises lors de la résolution des problèmes; ceci, joint au succès au problème 1 prouve qu'une certaine compréhension des fonctions sur les chaînes de caractères a été acquise. Nous avons fait la remarque que ce succès s'appuie très fortement sur une situation rencontrée antérieurement (lors de la gestion de la file d'attente dans le problème **FILE**): Natacha a reconnu des situations isomorphes, du point de vue des fonctions à construire, et a transféré directement la syntaxe employée dans la résolution du problème **FILE**; elle a certainement utilisé pour cela la possibilité de consulter les notes de cours, comprenant les programmes réalisés en classe. Des difficultés demeurent:

- l'oubli de l'affectation concernant la variable `L`; ceci peut être une conséquence de la façon dont nous avons présenté beaucoup de problèmes: les constantes du problème reçoivent une valeur par affectation dans le début de programme proposé avec l'énoncé, et la nécessité de certaines affectations peut ne pas apparaître; on rapprochera ceci de la difficulté à obtenir la mise à jour de la variable `L` dans le problème **FILE**.
- l'itération: le progrès dans la compréhension des fonctions sur les chaînes de caractères, révélé par le succès au problème 1 ne se confirme pas ici; l'élève était absente lors de la résolution de **ALPHA4** mais présente pour **PIANO1**, ces deux problèmes la même structure que le problème 3 de l'épreuve sur papier.
- la présence erronée des instructions `INPUT A$` et `INPUT B$` dans la réponse au problème 2. Nous avons pu constater que les débutants, faute d'une connaissance suffisante du dispositif, font de nombreuses erreurs d'interprétation concernant l'instruction d'entrée au clavier `INPUT`; nous préférons par conséquent attendre que les élèves aient une maturité suffisante, avant de leur faire rencontrer les difficultés liées à cette instruction; c'est pourquoi nous avons choisi d'éviter le plus possible que les élèves aient à employer cette instruction, sauf dans des cas très

simples (**INSEE3, FILE**). Nous ne disposons donc pas d'indication sur la façon dont cette élève interprète l'instruction d'entrée au clavier `INPUT`; il nous semble par contre que la présence erronée des instructions `INPUT A$` et `INPUT B$` peut s'expliquer par le fait que l'élève s'est aidé, pour construire sa réponse, de la liste du programme TESTVAL, introduit avant la résolution de **INSEE2** (chapitre 9 §3.4), pour vérifier la compréhension par les élèves de la fonction `VAL`; en effet, dans ce programme, les variables numériques `A$` et `B$` recevaient une valeur par entrée au clavier avant leur conversion en valeur numérique par la fonction `VAL`.

Karine

La réponse à la première question (**ONJOURB**) comporte l'erreur classique consistant à concaténer le premier caractère à droite de la chaîne, mais sans transformer la chaîne pour "ôter" ce premier caractère. On se souvient que le groupe **Karine-Natacha** avait buté, au cours de la première séance de résolution du problème FILE sur l'expression d'une chaîne privée de son premier caractère. La difficulté avait été résolue par Natacha à la séance suivante, mais **Karine** était absente. L'élève pense-t'elle que "l'extraction" du premier caractère, lors du calcul de `A$` a modifié `chaîne$`, et qu'en conséquence le calcul de `B$` est correct ? Nous faisions cette hypothèse en posant le problème au début de nos observations au cours de l'année 86-87. On remarquera cependant que jamais ce type de raisonnement n'a été exprimé par cette élève, même au cours de la période (relativement longue) où elle cherchait l'expression d'une chaîne privée de son premier caractère au cours de la résolution du problème **FILE**. Nous examinerons plus complètement cette question au paragraphe 7.

La seconde question (**SOMME**) est correcte.

La réponse à la troisième question (**CRYPTAGE**) ne comporte pas les avertisseurs d'itération; elle comporte cependant une instruction qui ne prend de sens que dans une itération (`LET X=X+1`). La ligne: `LET A$ = MID$(LETTRE$,X,1)` entraînerait une erreur pour `X` supérieur ou égal à 2 (`LETTRE$` est composé d'un seul caractère); nous avons rencontré dans l'observation d'autres cas où cette élève tente d'utiliser l'affectation et la fonction `MID$` en dehors de leur signification: par exemple `B$=MID$(A$,L-1,1)` pour exprimer la chaîne `LISTE$` "privée" du caractère `A$`. Ici l'expression pourrait vouloir dire: soit `X` le rang de `LETTRE$` dans `ALPHABET$`. Dans les réponses de **Karine**, les difficultés syntaxiques concernant l'affectation ont disparu; l'affectation semble ainsi mieux maîtrisée; mais l'expression `LET A$ = MID$(LETTRE$,X,1)`, où, pour l'élève, ce n'est pas `A$`, mais `X` qui prend une valeur , montre que des difficultés demeurent, en lien avec les difficultés d'emploi de la fonction sous-chaîne; ces difficultés apparaissent aussi bien dans un contexte non-itératif (problème 1) que dans un contexte itératif (problème 3); nous analyserons plus complètement cette difficulté au paragraphe 7.

Jérôme

La première question (**ONJOURB**) comporte l'erreur repérée chez **Karine** (concaténer le premier caractère à droite de la chaîne, mais sans transformer la chaîne pour "ôter" ce premier caractère). Le groupe **Jérôme-Arnaud** a rencontré les mêmes difficultés que le groupe **Karine-Natacha** pour obtenir l'expression d'une chaîne privée de son premier caractère; mais **Jérôme** était, lui présent au cours de la dernière séance de résolution qui a permis d'obtenir une expression satisfaisante pour cette question. Cet élève n'a pas, lui non plus exprimé au cours de la résolution des problèmes, l'idée que "l'extraction" du premier caractère d'une chaîne, modifie cette chaîne.

La seconde question (**SOMME**) est correcte.

La troisième question (**CRYPTAGE**) également, ce qui constitue une bonne surprise, compte tenu des difficultés rencontrées par cet élève par exemple dans **ALPHA4** qui a la même structure. Cet élève emploie une affectation à l'intérieur du corps de boucle, de façon que la condition d'arrêt comporte seulement un test d'égalité de deux variables. Nous avions noté que l'affectation à une variable du caractère courant au cours de l'itération complique la tâche de l'élève, puisqu'elle introduit une instruction de plus dans le corps de boucle, et donc des difficultés de séquentialité; ces

difficultés avaient été maîtrisées laborieusement par **Arnaud-Jérôme** lors de la résolution de **ALPHA3** et **ALPHA4**, alors que **Karine** employait la condition `LETTRE$=MID$(ALPHABET$,X,1)`; mais cette affectation se révèle ici utile puisqu'elle semble mettre **Jérôme** à l'abri de la confusion présente dans la réponse de **Karine** suite à l'écriture d'une condition trop complexe. De façon générale, il se confirme ici que, pour des élèves présentant des difficultés d'acquisition, l'affectation à des variables intermédiaires permet d'obtenir des écritures moins complexes; ces écritures seraient plus facile à interpréter; les élèves seraient moins tentés de les interpréter selon des conceptions erronées du dispositif. En contrepartie, les programmes comportent davantage d'instructions, ce qui peut entraîner des difficultés (surtout dans le cas d'une itération); mais la réussite de **Jérôme** et l'échec de **Karine** nous conduisent à penser que ces difficultés peuvent être plus facilement surmontées que les confusions qui proviennent d'écritures trop complexes.

Bilan des acquisitions des 3 élèves observés; comparaison avec EPR1 .

problème	résultat	élèves observés	classe 86-87 (EPR1)
ONJOURB *construction de fonction*	succès	1 / 3	5 / 11
	syntaxe incorrecte	0 / 3	5/ 11
	erreur sémantique	2 / 3	1 / 11
	utilisation de la concaténation	3 / 3	0 / 11
SOMME *choix type*	succès	3 / 3	0 / 11
	résolution avec erreur de type	0 / 3	1 / 11
	résolution avec erreur de type et erreur de calcul sur les ordinaux	0 / 3	2 / 11
	pas de programme	0 / 3	8 / 11
CRYPTAGE *construction itération*	succès	1 / 3	2 / 14
	pas d'itération	1 / 3	6 / 14
	itération erronée	1 / 3	2 / 14
	pas de programme	0 / 3	4 / 14

La comparaison entre les résultats des 3 élèves observés et la classe 86-87 est à prendre avec précautions; en effet cette classe comportait des élèves à l'aise en informatique, et d'autres ayant plus de difficultés, alors que les 3 élèves observés ont été choisis parmi ceux présentant les difficultés d'acquisition les plus importantes dès la phase d'alphabétisation. Les circonstances de passation sont comparables, mais le texte des problèmes 2 et 3 diffère d'une passation à l'autre. Dans le problème **ONJOURB**, le taux de succès des 3 élèves observés est plutôt inférieur à EPR1. Nous analyserons au paragraphe suivant cet insuccès comme provenant d'une difficulté insuffisamment prise en compte pour la construction des séries de problèmes. Par contre, les élèves utilisent correctement la concaténation et la notation fonctionnelle de la fonction `MID$`; dans EPR1, en dehors des élèves les plus à l'aise qui réussissent au problème, les erreurs proviennent surtout de confusions entre affichage et concaténation et de difficultés à prendre en compte la notation fonctionnelle. Dans le problème **SOMME**, le succès des 3 élèves observés, comparé à l'insuccès des élèves de la classe 86-87, montre que la difficulté *D.TYPE* peut être facilement résolue à l'aide de problèmes adaptés. Dans le problème **CRYPTAGE**, malgré l'utilisation d'une forme d'itération plus facile à mettre en oeuvre, seuls les élèves les plus brillants de la classe 86-87 réussissaient cet problème; le succès d'un des élèves observés montre que, même avec des élèves présentant a priori peu de dispositions, des progrès peuvent être observés, suite à la résolution de problèmes adaptés.

6. Récapitulation des difficultés rencontrées par les élèves

Le tableau suivant reprend l'observation et le dépouillement de l'épreuve sur

papier; il permet également de connaître la séance où le problème a été résolu, et la durée moyenne de cette résolution pour les élèves observés:

Problème et séance	Type de tâche	Natacha-Karine	Arnaud-Jérôme
ALPHA1 séance 1 durée 1/4h	calcul arguments.	Pas de difficultés. (Natacha absente séances 1 et 2)	
ALPHA2 séance 1 1/4h	construction fonctions.	Utilisation erronée de MID$. (Natacha absente séances 1 et 2)	
ALPHA3 séance 2 durée 3/4h	construction itération.	Affectation (incrémentation du compteur de boucle).	Affectation (variable intermédiaire, compteur de boucle).Séquentialité dans le corps de boucle.
ALPHA4 séance 2 durée 3/4h	construction itération, construction de fonctions	Difficultés: calcul itératif de la position, de la lettre donnée, séparation du calcul de la position et du calcul du résultat, confusion entre condition et affectation.	Recherche abrégée (manque de temps)
PIANO1 séance 3 durée 1h1/4	construction itération,.constructio n fonctions.	Condition d'arrêt, initialisation, évolution du compteur de boucle.	Position des avertisseurs. condition d'arrêt.
TESTC1 séance 4 durée1/4h	calcul arguments	Confusion sur la fonction longueur (Karine)	Pas de difficultés.
TESTC2 séance 4 durée 1/2h	calcul arguments	Rang de la lettre "milieu" calculée comme L/2 puis ajustée à L/2 + 0.5	Difficulté à concevoir que L/2 n'est pas le rang de la lettre milieu.
TESTC3 séances 4-5 durée 1h1/2	construction itération	Condition d'arrêt, évolution des paramètres de boucle, initialisation et séquentialité.	Erreur de type, évolution des paramètres de boucle, condition d'arrêt et séquentialité. (Arnaud absent séances 5 et 6)
INSEE1 séance 6 durée 3/4h	choix type.	Constantes chaînes.	Erreur déclaration de type. (Arnaud absent séances 5 et 6)
INSEE2 séance 7 durée 1h	choix type.	Affectation.	Calculs d'ordinaux.
INSEE3 séances 7-8 durée 1/2h	choix type.	Constantes chaînes, Confusion constantes variables.	Erreur déclaration de type.
FILE séances 8-9 durée 1h1/2	construction fonction.	Ecriture d'une expression "ôtant" le premier caractère d'une chaîne difficile à obtenir. Articulation d'expressions conditionnelles. (Karine absente séance 9)	

Epreuve sur papier: (durée maximum 1h).

question	Type de problème.	Natacha	Karine	Arnaud
n°1 (ONJOURB)	*construction fonction.*	Réussite.	Erreur compréhension fonction MID$	
n°2	*choix type.*	réussite.		
n°3 (CRYPTAGE)	*construction itération.*	Pas d'itération; erreur de type.	Absence d'avertisseur d'itération, difficulté avec l'affectation.	Réussite.

7. analyse par type de difficulté et conclusions

7.1. La difficulté à envisager le type des objets en fonction des opérations nécessitées par le problème, et non en fonction de leur aspect externe (*D.Type*).

7.1.1 Les élèves ayant travaillé sur la progression proposée au chapitre précédent, surmontent cette difficulté

Nous avions constaté lors de la passation de l'épreuve 1 en 86-87 que les élèves confrontés à une question simple présentant cette difficulté, omettaient tous de déclarer comme chaîne les variables sur lesquelles ils faisaient porter les fonctions "chaînes"; et dans d'autres problèmes, les erreurs provenant d'une confusion de type (numérique-chaîne) étaient nombreuses. La série **INSEE** présentée au chapitre précédent a été construite en fonction de ces difficultés; elle propose aux élèves une situation où des quantités numériques doivent être représentées par des chaînes; elle introduit une fonction de conversion (chaîne → numérique) et donne des occasions de l'utiliser. Nous constatons, au cours de la résolution de cette série que les difficultés attendues sont rencontrées et progressivement résolues; une difficulté supplémentaire est rencontrée avec l'écriture des constantes chaînes: des confusions apparaissent: confusion avec l'écriture des identificateurs de variables chaînes (signe $ à la fin de la constante au lieu des guillemets), confusion dans la concaténation de variables chaînes et de constantes. Le succès général à la question 2 de l'épreuve sur papier atteste que les difficultés attendues ont été résolues: les élèves déclarent le type des variables selon les fonctions qu'ils font opérer, et emploient correctement la fonction de conversion.

7.1.2 Quel est l'influence de ce progrès sur la conception que les élèves se font des objets informatiques ?

Une de nos hypothèses est que le travail sur les types doit permettre aux élèves une meilleure conception des objets, et les aider ainsi à progresser sur les difficultés plus générales qu'ils rencontrent dans leurs apprentissage en informatique. Les erreurs de type ne sont pas de simples erreurs de syntaxe, mais sont liées à la façon dont les élèves considèrent les objets informatiques ainsi que le montre l'exemple suivant: l'addition `A$ + N` est conçue par l'élève qui l'emploie comme une translation faisant passer du caractère `A$` d'une chaîne `CHAINE$` au caractère dont le rang dans `CHAINE$` est la somme du rang de `A$` et de `N`; elle se rencontre sous la forme `MID$(CHAINE$, A$ + N, 1)` , ou, de façon plus concise `A$ + N`, la variable `CHAINE$` étant alors implicite pour l'élève; cette addition est donc liée à la confusion entre un ordinal et le caractère dont il est la position dans une chaîne; l'absence de distinction des types des objets est un des éléments de cette confusion, et l'on peut penser qu'un travail sur le type des objets permettra aux élèves de comprendre que ce type d'écriture n'a pas de sens.

Confrontons cette hypothèse à l'observation des phases où apparaissent des erreurs de type. Les premiers problèmes (série **ALPHA** et **PIANO**) comportent de nombreuses écritures où les objets ont un statut très flou; nous indiquons l'écriture en cause, suivie du titre du problème, du nom de l'élève, et d'une interprétation de

l'écriture:

LETTRE$="MAJUSCULE"	**ALPHA2 Karine**	pour indiquer que le caractère contenu dans LETTRE$ est une Majuscule
LETTRE$=MID$(ALPHABET$,N,MAJUSCULE)	**ALPHA2 Karine**	pour indiquer que le caractère contenu dans LETTRE$ est une Majuscule en précisant que c'est la lettre de rang N.
LET LETTRE$=X	**ALPHA3 Karine**	pour indiquer que LETTRE$ est le Xième caractère), la chaîne étant implicite)
RANG$=MID$(GAMME$,X,1)	**PIANO1 Karine-Natacha**	RANG$ n'a pas de signification comme identificateur de variable; l'élève veut seulement signifier que X doit prendre comme valeur le rang de LETTRE$ dans GAMME$); ici il n'y a plus d'erreur de syntaxe, mais la confusion demeure.
LET NOMBRE$= X + 1	**ALPHA3 Arnaud-Jérôme**	pour incrémenter X; comme RANG$ plus haut pour **Karine**, NOMBRE$ n'a pas de signification dans le problème
DO LET A$= A-X LET B$= A+X LOOP UNTIL A$=L AND B$=L	**TESTC3 Jérôme-Michaël**	ces élèves construisent une boucle où A$ et B$ parcourent la chaîne, en partant de la position médiane, repérée par la variable A, et en restant symétriques).

La série **INSEE** est conçue pour placer les élèves devant le choix d'un type de donnée non numérique pour représenter des données d'aspect externe numérique; il n'est donc pas étonnant de trouver des erreurs de type dans les écritures que produisent les élèves au cours de leur recherche; ces erreurs concernent dans le premier problème les identificateurs (**Arnaud-Jérôme**) et les constantes (**Natacha-Karine**); mais, ensuite, les élèves intègrent sans grande difficulté le fait que le résultat d'une fonction MID$ est une chaîne (donc doit être affecté à une variable dont l'identificateur se termine par un $), utilisent aisément la fonction de conversion VAL, et distinguent addition et concaténation; seule l'expression des constantes leur pose un problème.

Dans la dernière série (**FILE**),(après la série **INSEE**), des écritures comportant des erreurs de type sont facilement remises en cause par le groupe **Natacha-Karine**, comme, par exemple LISTE$= LISTE$ - 1 (pour "ôter" le premier caractère de LISTE$). De même, et dans le même but, **Arnaud-Jérôme** proposent spontanément l'écriture LET LISTE$=LISTE$-MID$(LISTE$,1,1), mais ils acceptent facilement que l'opération de signe - n'ait pas de sens dans les chaînes.

Les réponses de **Karine** et **Jérôme** à l'épreuve sur papier ne comportent aucune erreur de type, ou erreur qui viendrait d'une confusion concernant le type des objets. On trouve chez **Natacha** dans l'épreuve sur papier une addition entre caractère et ordinal, sous une des formes décrites ci-dessus: LETTRE1$ = MID$(ALPHABET$,LETTRE$ + 1,1) (pour affecter à LETTRE1$ le caractère suivant LETTRE$ dans la chaîne ALPHABET$). A l'issue de cette série, on peut donc penser que **Karine** et **Jérôme** ont réellement progressé vers une meilleure compréhension des objets; cette meilleure compréhension était en germe après les séries **ALPHA**, **PIANO** et **TESTCHA**; elle s'est trouvée confirmée dans la série **INSEE**; la difficulté semble demeurer chez **Natacha**, ce qui peut être une conséquence de son absence lors des premières séances. Ainsi, le travail spécifique sur les types, au cours de la résolution

de la série **INSEE**, viendrait plutôt consolider une conception en cours d'apparition chez **Karine** et **Jérôme**, et se révélerait moins fécond chez **Natacha**, qui, absente aux séances 1 et 2, aborde ce travail pour sa quatrième séance.

7.1.3 La question des opérateurs de comparaison

Les opérateurs de comparaison à valeurs booléennes (supérieur et inférieur) ont quand ils portent sur les chaînes , la signification d'une comparaison suivant l'ordre alphanumérique, l'ordre sur les caractères étant donné par le codage interne de ces caractères; cet ordre est compatible avec l'ordre alphabétique pour les lettres, mais s'étend à tous les caractères considérés par l'ordinateur, en particulier les chiffres (zéro à 9) (ceci est vrai dans les langages évolués BASIC , LSE, PASCAL, **LOGO** ...). Lorsque les chaînes sont de même longueur et sont constituées de chiffres, la comparaison à l'aide d'un de ces opérateurs donne le même résultat que la comparaison des nombres dont l'écriture en base 10 coïncide avec ces chaînes: `"123" > "456"` a la même valeur que `123 > 456`. Cette particularité a permis aux élèves de contourner des difficultés dans le problème **INSEE1**; il s'agissait dans cet problème de vérifier que le premier chiffre d'une chaîne composée de 13 chiffres est 1 ou 2 et que les 3ème et 4ième chiffres forment le numéro d'un mois de l'année.

Par contre `"123" > "45"` est faux alors que `123 > 45` est vrai. C'est pourquoi nous regrettons a posteriori de ne pas avoir introduit un problème mettant en cause la conception selon laquelle les opérateurs donneraient systématiquement le même résultat, que les objets soient numériques ou de type chaîne; en effet, cette conception, sur laquelle les élèves se sont appuyés pour leur anticipation, est basée sur l'idée que même déclarée comme chaîne, une écriture composée de chiffres reste un nombre, sur lequel des opérateurs numériques de comparaison peuvent valablement opérer.

7.2. Les difficultés avec les éléments généraux du langage (*D.Langage*)

7.2.1 L'affichage des chaînes; la confusion entre calcul et affichage

Arnaud-Jérôme rencontrent une difficulté concernant l'affichage des chaînes dans **INSEE3**; ils ont dans cet problème à former le numéro INSEE d'une personne après avoir entré les renseignements le concernant; ils tentent d'afficher, en une seule instruction plusieurs variables-chaînes, suivies d'une constante; cet affichage doit suppléer au calcul du numéro INSEE, qui utiliserait la concaténation, et que les élèves n'envisagent pas spontanément. L'affichage, en une seule instruction de plusieurs variables- chaînes, suivies d'une constante est une construction possible; mais **Arnaud-Jérôme** maîtrisent mal les écritures respectives des identificateurs de variables chaînes et des constantes chaînes. Il est vrai qu'à ce moment de la progression, la concaténation est peu disponible pour ces élèves; ils restent donc sous l'influence des affichages réalisés lors des séances qui ont précède l'apprentissage des chaînes de caractères; il est courant, en effet, dans les programmes manipulant des données numériques, de faire intervenir des constantes chaînes lors d'affichage de résultats numériques, afin de rendre le message plus explicite: exemple `PRINT "Votre age est de "; AGE`. L'indication que nous leur donnons est de *calculer* d'abord le résultat puis de l'afficher; cette indication est cohérente avec la forme de programmation que nous pensons la plus efficace pour ces élèves: un calcul produisant un résultat qui est affecté à une variable avant d'être affiché; l'efficacité d'une telle forme se confirme ici puisque les élèves terminent seuls le programme. Les 4 élèves observés ne rencontrent pas ailleurs des difficultés qui résulteraient de l'affichage des chaînes, ou d'une confusion entre l'affichage et le calcul.

A l'épreuve finale, seul **Michaël** fait une erreur de ce type. Dans le problème 1 il isole le premier caractère par un affichage `PRINT MID$(CHAINE$,1,1)`; ensuite, le reste de la chaîne est isolé par une affectation `LET CHAINES=MID$(CHAINE$, 2, L)`. Il est certain, d'une part que la concaténation n'est pas acquise par cet élève, d'autre part, que l'instruction d'affichage est là essentiellement pour que `MID$(CHAINE$,1,1)` ne soit pas seule dans l'instruction (**Michaël** a pu remarquer qu'une telle expression comme instruction isolée produit une erreur); la notion de calcul

sur les chaînes n'est pas acquise; elle n'est pas distinguée de l'affichage. Dans le même problème, **Natacha** calcule correctement le résultat, mais omet de l'afficher. Cet oubli pourrait être une manifestation de la confusion calcul-affichage, l'élève considérant qu'un résultat calculé est automatiquement porté à la connaissance de l'utilisateur; peut-être au contraire, l'élève considère-t'elle que l'essentiel est le calcul, et qu'elle a donc répondu à la question. Cet oubli nous paraît sans grande importance: la nécessité de calculer étant acquise, l'absence d'affichage lors de l'exécution conduira l'élève à compléter son programme.

7.2.2 L'affectation

7.2.2.1 Remarques préliminaires sur l'emploi de l'affectation par les élèves

Les élèves observés utilisent largement l'affectation pour les résultats intermédiaires; nous les y encourageons tant que la notation fonctionnelle n'est pas suffisamment intégrée. En effet, nous avons vu au chapitre **6** que les élèves qui utilisent de façon systématique l'affectation à des variables intermédiaires pour décomposer des expressions fonctionnelles complexes, déjouent plus facilement les difficultés propres à l'écriture des fonctions. De façon à distinguer calcul et affichage, nous demandons également aux élèves observés d'affecter le résultat du calcul faisant l'objet du problème à une variable avant de l'afficher (voir chapitre précédent paragraphe 2.1). Les élèves produisent donc beaucoup d'écritures où l'affectation est présente; comme la lecture des comptes rendus des séances de résolution permet de le constater, de nombreuses écritures où l'affectation est présente sont erronées; nous allons les analyser au prochain paragraphe.

On pensera peut-être que c'est parce-qu'ils utilisent abondamment l'affectation que les élèves font beaucoup d'erreurs, et qu'en recourant moins, voire pas du tout à l'affectation, donc en adoptant d'emblée un style de programmation "fonctionnel", ils éviteraient ces nombreuses erreurs. Nous nous appuyons sur l'épreuve et les entretiens analysés au chapitre **7** pour penser que c'est la représentation que se font les élèves du dispositif qui est en cause, et non le style de programmation. Rappelons l'exemple suivant vu au chapitre **6**: à la question "Ecris un programme pour que l'utilisateur, ayant entré une chaîne au clavier, l'ordinateur affiche la chaîne en passant la première lettre à la fin"; la réponse d'une élève est la suivante: `LET A$=MID$(X$,2,L+MID$(1,1))`. Cette écriture comporte une seule affectation, là où d'autres élèves en écrivent trois (une pour le calcul du premier caractère, la seconde pour le calcul du "reste de la chaîne", et la troisième pour la concaténation des deux premiers résultats); l'écriture ci-dessus n'a pas de sens dans le langage de programmation. Elle a, par contre, une signification pour l'élève, que nous avons analysée comme la compréhension de la notation fonctionnelle comme codage d'action *(D.Langage).* Tentant d'"améliorer" avec l'élève, cette écriture sans la séparer, nous n'avons pas pu obtenir une meilleure compréhension de la part de l'élève.

7.2.2.2 les difficultés dans l'utilisation de l'affectation, et leur évolution

La représentation de l'affectation chez **Karine:**

Le tableau récapitulatif (paragraphe 6), indique que chez une élève (**Karine**) les difficultés concernant l'affectation s'observent tout au long de la résolution des problèmes, et dans l'épreuve sur papier; analysons comment évoluent ses conceptions:
Lors de la résolution de **ALPHA3**, on rencontre:

- une affectation sans identificateur résultat ni signe "=": `LET X+1`

et dans **ALPHA4**

- une affectation ne respectant pas l'ordre identificateur, expression , et comportant une confusion de type: `LET LETTRE$=X` qui à pour sens "X est le rang de `LETTRE$`"
- une confusion entre affectation et condition d'arrêt: l'élève écrit `LOOP UNTIL LETTRE$=MID$(ALPHABET$, X + 26, 1)`, alors que la sortie de boucle doit se faire lorsque la donnée `LETTRE$` est la Xième lettre, et que, cette condition étant réalisée, le résultat s'écrit `MID$(ALPHABET$, X + 26, 1)`.

On retrouve une erreur de ce type dans **INSEE2**:

- `89-D = VAL(D$)`; pour indiquer que `D` doit prendre comme valeur l'age d'une

personne dont l'année de naissance (sous forme d'une chaîne de deux caractères) serait D$.

Dans la résolution de **FILE**,

- pour "ôter" de LISTE$, **Karine** propose de façon insistante l'écriture B$ = MID$(A$, L-1,1)

Dans l'épreuve sur papier, à la troisième question (**CRYPTAGE**)

- confrontée à la recherche de la position d'une occurrence du caractère LETTRE$ dans la chaîne ALPHABET$, l'élève écrit l'instruction suivante LET A$= MID$(LETTRE$,X,1)

Rappelons la forme prise par l'affectation dans les langages impératifs, avant d'examiner la représentation que se fait l'élève: une affectation comprend l'identificateur de la variable qui doit prendre une nouvelle valeur, puis un signe spécifique (= en BASIC ,:= en PASCAL, ← en LSE) suivi d'une expression qui est évaluée et dont la valeur est affectée à la variable dont l'identificateur précède le signe spécifique; cette expression peut, comme toute expression, faire intervenir des identificateurs de variable, qui sont remplacées par leur valeur au moment de l'évaluation. La variété de BASIC utilisée ici permet de placer le mot réservé LET avant une affectation, de façon à ce que le lecteur du programme distingue plus facilement l'affectation d'une condition utilisant l'opérateur de comparaison "égalité". Ainsi, l'affectation X=X+1 ou LET X=X+1 en BASIC , s'écrit en PASCAL: X:=X+1 , et en LSE: X← X+1 . Dans ces écritures, les deux symboles X ont des statuts différents: le symbole X avant le signe = doit être compris comme l'indication que la variable d'identificateur X doit évoluer, alors que celui qui suit doit être remplacé par la valeur actuelle de la variable.

Karine, quant à elle a une conception de l'affectation que l'on pourrait décrire ainsi: une affectation sert à indiquer à l'ordinateur que l'on veut qu'une variable évolue; il suffit donc de "lui dire" comment cette évolution doit se faire, ou quelle condition doit être réalisée. Toute écriture sans ambiguïté pour l'élève peut convenir:

- LET X+1: ajouter 1 à X; il est clair ici pour l'élève, que c'est x qui doit évoluer, puisque c'est le seul identificateur en présence).
- LET LETTRE$=X: X doit prendre pour valeur la position de LETTRE$ dans la chaîne ALPHABET$; il est clair, à cet endroit du programme, que X doit recevoir une valeur, puisque LETTRE$ a déjà une valeur.
- 89-D = VAL(D$) pour affecter la variable D: il est remarquable que cette difficulté n'est pas apparue depuis plusieurs séances et ressurgit ici à l'occasion de l'utilisation d'une fonction nouvelle (VAL, fonction de conversion de type qui à une chaîne composée de chiffres fait correspondre le nombre dont cette chaîne est l'écriture décimale).
- Dans la résolution de **FILE**, ayant affecté à A$ le premier caractère de LISTE$, les élèves ont à "ôter" de LISTE$ ce premier caractère, ce qui constitue pour elles une difficulté majeure; **Karine** propose de façon insistante l'écriture B$ = MID$(A$, L-1,1); cette expression est contestée à juste titre par Natacha qui fait remarquer que "A$ c'est une seule lettre". B$ n'a pas de signification dans le problème; il n'est pas question, pour Karine, de l'affecter ensuite à une variable du problème telle que LISTE$; l'affectation (syntaxiquement correcte) sert, dans la compréhension qu'en a l'élève, à éviter l'erreur que produirait la présence isolée de MID$(A$, L-1,1); mais pour l'élève, c'est bien cette dernière expression qui doit modifier LISTE$, en lui "ôtant" A$; l'élève n'exprime pas verbalement sa compréhension de cette expression; mais on peut l'interpréter comme "extraire A$, de façon que la longueur soit L-1...".
- Une affectation étant comprise comme une écriture qui doit être réalisée après l'exécution de l'instruction, il n'est pas étonnant qu'il y ait, dans **ALPHA4**, confusion avec la partie condition d'une expression conditionnelle (et en particulier, avec la condition d'arrêt d'une itération).
- Dans l'épreuve sur papier, la même conduite s'observe dans la troisième question: confrontée à la recherche de la position d'une occurrence du caractère LETTRE$ dans la chaîne ALPHABET$, l'élève ne parvient pas à construire une itération, et écrit l'instruction suivante: LET A$= MID$(LETTRE$,X,1). Comme B$ plus haut, l'identificateur A$ a seulement une utilité pour la correction syntaxique;

l'élève veut signifier que X doit prendre comme valeur la position du caractère LETTRE$, la chaîne ALPHABET$ étant implicite; on retrouve le type de représentation erronée qui a conduit à l'écriture LET LETTRE$=X dans ALPHA4; mais depuis, l'élève a travaillé sur les types et sait que cette dernière expression est incorrecte.

On remarquera que, dans le cas où l'expression à affecter ne dépend pas de la variable affectée, la conception correcte de l'affectation est "compatible" avec cette représentation, en ce sens qu'elle en est un cas particulier: le cas où seule la variable dont l'identificateur précède le signe = évolue; il en résulte que la représentation erronée est difficile à mettre en échec puisque le succès enregistré par l'élève lors de l'utilisation d'une forme correcte peut confirmer chez elle la représentation erronée.

Rapports entre la représentation de l'affectation chez **Karine** et l'écriture de clauses en programmation déclarative.

La représentation de **Karine** conduit à une forme d'expression que l'on pourrait qualifier d' "incantatoire": il s'agit d'un commandement ou d'une "prière" qui ne prend pas en compte la nécessité d'indiquer au dispositif les moyens de réaliser cette "prière"[120]. Nous avons remarqué dans l'analyse de la séance 7, qu'une telle forme d'expression pourrait être rapprochée des clauses énoncées dans une forme de programmation appelée déclarative et dont le langage le plus représentatif est **PROLOG.**

Ainsi, le prédicat date_naissance(personne, DATE) reliant le nom de chacune des personnes d'une base à sa date de naissance étant défini, si l'on veut calculer l'age au premier janvier 89 d'une personne, on pourra, en programmation déclarative écrire la clause:

```
age(PERSONNE, D) si date_naissance(PERSONNE,DATE) et 88-D= DATE.
```

et obtenir l'age d'une personne nommée Alfred par la requête

```
quel D age("Alfred",D)
```

On constate sur cet exemple la similitude de l'écriture de cette clause avec la forme employée ci-dessus par **Karine**. Peut-on en déduire que la programmation déclarative serait mieux adaptée compte-tenu des difficultés de cette élève? Nous ne le pensons pas; en effet, la clause ci-dessus n'est possible que parce que le langage comporte un algorithme pour la fonction inverse de la fonction D →88 -D, et parce que la première partie de la clause indique que, PERSONNE étant connue, c'est D qui doit prendre une valeur. Par contre, si en **PROLOG**, on demande à un élève de construire une clause racine_carre(NOMBRE, X) qui relie à NOMBRE le nombre positif dont il est le carré, l'élève pourra être tenté d'écrire la clause suivante:

```
racine_carre(NOMBRE, X) si X>= 0 et X*X = NOMBRE
```

Cette clause échouera si, comme c'est probable, le langage ne dispose pas d'un algorithme de recherche d'une racine carré; après cet échec, l'élève devra construire un algorithme dont le principe sera le même que dans un langage algorithmique, même si son écriture doit être adaptée aux contraintes du langage déclaratif. Donc pour programmer vraiment en déclaratif, il est indispensable de prendre en compte les contraintes du dispositif, en particulier l'indication de la variable qui doit recevoir une valeur, et l'existence d'un algorithme permettant de calculer cette valeur; ces contraintes prennent en déclaratif une coloration différente, puisque la(les) variable(s) qui reçoivent une valeur sont les variables libres de l'expression, et que le plus souvent, plutôt que d'implémenter un algorithme, on essayera de tirer le meilleur parti des algorithmes généraux (en particulier l'algorithme d'effacement des buts) implémentés dans le langage[121].

La facilité avec laquelle les premières clauses peuvent être obtenues dans une

---

[120] Selon le dictionnaire Le PETIT ROBERT, une *incantation* est «*l'emploi de paroles magiques (...) pour opérer un charme, un sortilège*». Le dictionnaire cite BERGSON : «*L'incantation peut participer à la fois du commandement et de la prière*».

[121] Voir par exemple : [GIANESINI & a.85] F.Gianesini H.Kanaoui R.Pasero M.Van Caneghem PROLOG InterEditions 1985

initiation à la programmation déclarative est trompeuse, car ces clauses ne sont possibles que par l'existence d'algorithmes de recherche dans des ensembles finis (algorithme d'effacement), et de certains algorithmes de calcul des fonctions inverses de fonctions arithmétiques, ou sur les chaînes. L'acquisition de compétences réelles en programmation déclarative pourrait donc par la suite, se heurter à des difficultés importantes, même si au premier abord, les clauses paraissent plus proches de la forme spontanée d'expression d'une élève comme Karine[122].

Y a-t'il chez **Karine** une progression vers une meilleure acquisition ?

A la lecture de l'analyse précédente, on constate que la représentation erronée de l'affectation persiste tout au long des apprentissages; l'affectation évolue en premier lieu syntaxiquement, en ce sens qu'on ne rencontre plus l'omission de l'identificateur de variable précédant le signe = , à partir de la troisième séance; mais cet identificateur peut être placé par l'élève pour la seule correction syntaxique et donc, ne pas avoir de signification; de même, les écritures impliquant une erreur de type disparaissent après la résolution de la série INSEE, mais l'élève compense par l'utilisation (erronée) de `MID$`, qui renvoie à une incompréhension de cette fonction. Ces remarques mettent l'accent sur la persistance de difficultés chez cet élève; néanmoins, le bilan des acquisitions de cette élève dans le domaine de l'affectation est assez positif si l'on en juge par l'épreuve sur papier; elle y emploie 12 fois l'affectation; les 12 écritures sont syntaxiquement correctes, et ne comportent pas d'erreur de type; seule une de ces affectations (analysée ci-dessus) traduit une représentation erronée. Cette erreur survient alors que l'élève est face à une difficulté majeure: la recherche de la position d'un caractère dans une chaîne.

On peut donc raisonnablement penser que cette élève a progressé au cours des séances; à ce stade, les interventions du pédagogue auprès de cette élève pourraient se centrer sur les statuts respectifs des identificateurs de variables, de part et d'autre du signe = , en faisant remarquer que seule la variable dont l'identificateur précède le signe = voit sa valeur évoluer. Nous notons au passage que notre interprétation des difficultés de **Karine** n'est possible que par l'accumulation d'un nombre important d'observations d'écritures erronées; un tel diagnostic, et l'intervention pédagogique qui en découle ne sauraient être obtenus sur un nombre trop limité d'observations; ceci est à prendre en compte, aussi bien pour la pratique enseignante habituelle, que pour la conception de systèmes d'aide individualisée à l'élève portant un diagnostic en fonction des écritures produites par l'élève; cette observation sera reprise et développée au paragraphe 8.

Les difficultés des autres élèves:

Nous avons porté l'attention sur **Karine**, car ses difficultés concernant l'affectation persistent; chez d'autres élèves, les mêmes difficultés apparaissent, puis disparaissent; c'est particulièrement le cas pour **Arnaud-Jérôme** dans **ALPHA3** et **PIANO1**:

- Confusion entre affectation et condition: `LET LETTRE$ = MID$(ALPHABET$,1,X)` dans ALPHA3
- Affectation sans signe =: `LET X+1` également dans ALPHA3; correction syntaxique de cette expression , sans évolution de la signification: `LET NOMBRE$=X+1`, au lieu de `LET X=X+1`.
- Confusion entre affectation et condition: `LET NOTE$ = MID$(GAMME$,X+3,1)` dans **PIANO1**

Puis cette difficulté disparaît; les réponses de **Jérôme** à l'épreuve sur papier ne font apparaître aucune erreur de ce type, de même pour **Natacha**. Ces élèves ont donc également progressé dans l'acquisition de l'affectation.

7.2.2.3 le rôle des types de données dans cette évolution

Y a t'il dans cette évolution une influence de la fréquentation d'un type non-numérique? Ou le seul fait de l'activité de résolution a t'il permis ce progrès,

---

[122][TAYLOR du BOULAY 87] Learning and using Prolog An empirical investigation.Cognitve Science Research Reports n°90. brighton U. of Sussex , cité dans [ROGALSKI 91]

indépendamment du type de donnée manipulé ?

Le nombre de séances précédant les 9 séances où nous sommes intervenu, est d'environ 30; ces séances ont consisté essentiellement en la résolution de problèmes portant sur des quantités numériques. Pourtant, au début des 9 séances, des affectations telles que `LET X+1` sont nombreuses, puis disparaissent progressivement. Au cours de ces 9 séances, les élèves observés ont certes bénéficié d'un encadrement renforcé; mais ce nombre de séances est relativement faible par rapport aux 30 séances précédentes; ainsi, le seul fait d'avoir mené une activité de résolution impliquant l'emploi de l'affectation ne nous semble pas suffisant pour expliquer les progrès concernant l'affectation.

Nous avons souligné au paragraphe concernant l'acquisition des types (**8.1.2**) à quel point la représentation des objets est floue chez les élèves lors des premières séances; dans une écriture comme `LET LETTRE$="Majuscule"` , l'élève ne conçoit pas l'écriture `"Majuscule"` comme représentant un objet (une constante chaîne), mais comme un "message" à envoyer à l'ordinateur pour qu'il comprenne que l'on veut que `LETTRE$` ait comme valeur un caractère en Majuscules. Ainsi les élèves doivent ils passer d'une représentation de l'affectation comme forme linguistique assez générale impliquant une évolution des données, à une représentation où des objets, représentés de façon formelle dans le dispositif, reçoivent des valeurs. On peut donc penser que c'est bien le travail sur les types de données qui a entraîné chez les élèves une meilleure représentation des objets manipulés par le dispositif, et en conséquence une meilleure compréhension de l'affectation comme processus intervenant sur ces objets.

7.3. Les difficultés liées à l'acquisition de la logique interne de la structure de chaîne (*D.Sem.Cha.*)

7.3.1. la concaténation

Au moment où elle est introduite, elle entre en conflit avec un moyen dont les élèves disposent pour obtenir que les valeurs de variables diverses apparaissent concaténées: l'affichage sans passage à la ligne; l'acquisition de la concaténation est donc liée à la conception du calcul comme indépendant de l'affichage; nous avons montré au chapitre précédent qu'une représentation naïve du calcul ne sépare pas ces aspects, mais que cette représentation ne peut suffire pour programmer. L'utilisation correcte de la concaténation par **Natacha, Karine, Jérôme** dans le problème 1 de l'épreuve sur papier pour calculer le résultat demandé avant de l'afficher, montre que la concaténation, à partir du moment où elle a été clairement séparée de l'affichage, ne pose pas de problème particulier d'acquisition.

Nous avons analysé en 8.2.a la réponse de **Michaël** au problème 1 de l'épreuve sur papier, et montré que la notion de calcul sur les chaînes n'est pas acquise par cet élève; cet élève n'emploie pas la concaténation, ce qui confirme que l'acquisition de la concaténation est liée à l'acquisition de la notion de calcul sur les chaînes.

7.3.2. la fonction sous-chaîne

7.3.2.1 L'acquisition du caractère fonctionnel des expressions utilisant MID$, et des contraintes syntaxiques qui y sont liées

Les élèves doivent s'obliger à insérer les écritures utilisant `MID$` dans un affichage, une expression, ou, le plus souvent, une affectation; selon les élèves et les circonstances, la prise en compte de cette contrainte syntaxique est liée de façon différenciée à l'acquisition du caractère fonctionnel de `MID$`: en effet, l'insertion par l'élève, de `MID$` dans un affichage, une expression, ou une affectation peut avoir pour seul but d'éviter une erreur de syntaxe; dans ce cas, l'écriture `MID$` (suivie de ses arguments), se suffit à elle même, et l'affichage, l'expression ou l'affectation n'ont pas de signification particulière pour le problème.

De façon à faire un bilan sur cette question, considérons les réponses des élèves à l'épreuve sur papier:

- **Karine** emploie 5 fois la fonction `MID$`; dans les 5 cas, `MID$` est insérée dans une affectation, de façon syntaxiquement correcte; dans 4 cas, l'identificateur précédant le signe = donne à l'affectation une signification correcte dans le problème; dans un

cas, cet identificateur n'a manifestement pas de signification: il s'agit au problème 3 d'affecter à X la position de LETTRE$ dans la chaîne ALPHABET$; l'élève écrit LET A$=MID$(LETTRE$,X,1); ici, pour l'élève, l'écriture MID$(LETTRE$,X,1) se suffit à elle-même, et l'écriture LET A$= est placée par l'élève pour la correction syntaxique.

- **Michaël** emploie 6 fois la fonction MID$, une fois dans une instruction d'affichage, quatre fois dans une affectation, et une fois dans une écriture qui est syntaxiquement une affectation, mais a en fait, pour l'élève, manifestement le sens d'une expression conditionnelle. L'affichage PRINT MID$(CHAINE$,1,1), comme première ligne de la réponse au problème 1, fait afficher le premier caractère de CHAINE$; cet affichage n'a pas de sens dans le problème, puisque ce caractère doit apparaître dans le résultat APRES le reste de la chaîne; (le programme doit permettre que, l'utilisateur ayant entré une chaîne au clavier, l'ordinateur affiche la chaîne en passant la première lettre à la fin); en fait il s'agit pour l'élève d'"extraire le premier caractère"; l'expression MID$(CHAINE$,1,1) se suffit à elle-même, et l'instruction d'affichage est placée uniquement pour la correction syntaxique[123].
- dans les réponses des 3 autres élèves, les écritures utilisant MID$, ne traduisent pas d'incompréhension du caractère fonctionnel de cette expression.

Nous retiendrons donc que les contraintes syntaxiques ont été intégrées; que cette intégration s'accompagne généralement d'une compréhension du caractère fonctionnel de MID$; mais que chez certains élèves, dans des circonstances particulières, le caractère fonctionnel peut être "oublié", la syntaxe restant néanmoins respectée.

7.3.2.2 Compréhension du rôle des arguments

- Le premier argument de MID$ est une chaîne: c'est la chaîne à partir de laquelle on obtient une sous-chaîne comme résultat de MID$

Seule parmi les élèves observés, **Karine** produit des écritures où le premier argument est erroné, la chaîne dont le résultat est "extrait" étant alors, pour elle, implicite:

B$ = MID$(A$, L-1,1)	dans la résolution du problème **FILE** a pour fonction d' "ôter" de la chaîne (implicite) LISTE$ le caractère A$;
LET A$=MID$(LETTRE$,X,1)	dans la résolution de **CRYPTAGE** (épreuve sur papier), a pour fonction d'affecter à X la position de LETTRE$ dans la chaîne (implicite) ALPHABET$.

Nous avons analysé plus haut ces écritures comme liées aux difficultés de l'élève concernant l'affectation, et à l'absence de compréhension de MID$ comme fonction rendant un argument; en effet, dans la compréhension de l'élève, les écritures MID$(A$, L-1,1) et MID$(LETTRE$,X,1) se suffisent à elles-mêmes; l'identificateur de variable, et le signe = qui les précédent ne sont là que pour la correction syntaxique. Il est clair que l'élève ne peut envisager comme premier argument de la fonction une chaîne autre que l'objet qui lui semble définir le problème: le caractère à "ôter" pour **FILE**, le caractère dont on cherche la position pour le problème **CRYPTAGE** (épreuve sur papier); il y a là une difficulté à changer de point de vue, à quitter le point de vue du but à atteindre pour adopter celui des moyens de l'atteindre, et donc des contraintes du langage.

Michaël (qui ne fait pas partie des élèves observés) produit également dans la résolution du problème **CRYPTAGE**, une écriture montrant une compréhension erronée du premier argument:

LET LETTRE$ = MID$(LETTRE1$,L,-1)	pour signifier que LETTRE$ doit être le caractère précédant LETTRE1$ dans la

[123] Cette interprétation peut paraître s'appuyer sur des indices ténus (une seule écriture). Nous pensons pouvoir la faire avec de bonnes chances de ne pas se tromper, grâce aux observations d'autres élèves (chapitre 6) manifestant de façon cohérente des conduites de ce type, et aux observations de cet élève en situation de résolution (ce chapitre).

chaîne (implicite) ALPHABET$

il y a également dans cette écriture une confusion entre affectation et condition, ainsi qu'une incompréhension du rôle des arguments numériques.

- Le second argument de MID$ est l'ordinal, position du premier caractère de la sous-chaîne résultat dans la chaîne argument; les énoncés mettent le plus souvent l'accent sur ce second argument, car il pose les difficultés du calcul sur les ordinaux; nous avons noté dans **ALPHA2** que le troisième argument (qui doit indiquer le nombre de caractères de la sous-chaîne) peut être compris comme une sorte de commentaire, corrigeant le second argument; dans ce cas, le couple (second argument , troisième argument) serait compris comme un tout; ce couple aurait pour fonction de donner toutes les informations nécessaires au calcul de la sous-chaîne; on remarquera une fois de plus que la conception correcte est "*compatible*" avec cette représentation erronée. Cette représentation erronée ne réapparaît pas dans les problèmes ultérieurs, chez les élèves observés. Elle apparaît chez **Michaël** dans la résolution de **CRYPTAGE** (épreuve sur papier).

7.3.2.3 La difficulté à calculer une partie de la chaîne définie comme un "reste" et la représentation des objets comme entités physiques non duplicables.

Cette difficulté apparaît dans **FILE** pour tous les élèves observés lorsqu'il s'agit de calculer la chaîne LISTE$ privée de son premier élément; nous devons, dans les deux groupes, insister sur le fait que la fonction MID$ suffit; les écritures:

MID$(A$,L-1,1) où A$ représente le premier caractère (**Karine**)

MID$(LISTE$,1,L-1) (**Natacha**)

montrent la difficulté à passer du point de vue de "ce qu'on ôté" (point de vue qui est celui de la question posée), au point de vue de "ce qui reste" (point de vue qu'il est nécessaire d'adopter pour utiliser la fonction sous-chaîne). On remarquera également que, comme au chapitre 6, l'expression L-1, qu'elle soit en second ou troisième argument code l'action d'ôter un caractère.

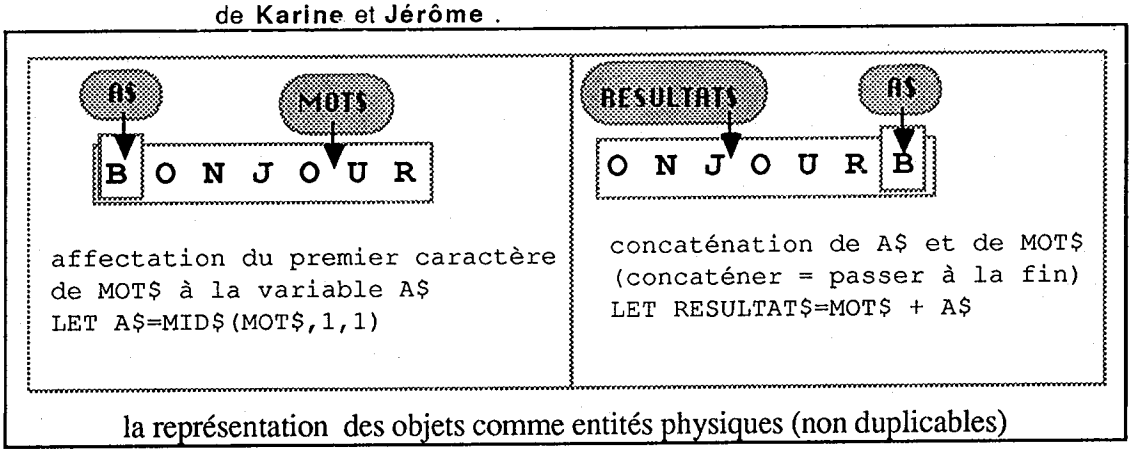
La difficulté se rencontre à nouveau dans le premier problème de l'épreuve sur papier; **Natacha** répond correctement, en utilisant le schéma de résolution du problème **FILE**; **Karine** et **Jérôme** ne calculent pas la chaîne CHAINE$ privée de son premier élément; ils se contentent de concaténer dans l'ordre CHAINE$ et le premier caractère calculé antérieurement et affecté à une variable; l'hypothèse selon laquelle, pour certains élèves, l'emploi de la fonction MID$ modifie par effet de bord la chaîne argument se verrait ici vérifiée.

Nous avons cependant remarqué que cette conception n'a été explicitée ni par **Jérôme** ni par **Karine** au cours des séances de travail sur ordinateur, et en particulier au cours des séances 8 et 9, où ces élèves ont travaillé sur le problème **FILE**, dont un des modules imposait de supprimer le premier caractère d'une chaîne. Examinons les différences entre les deux situations de résolution, de façon à tenter d'expliquer cette différence de comportement:

- la situation pédagogique est différente: dans un cas, une résolution guidée par un enseignant, dans l'autre, un travail autonome; dans la résolution guidée , l'insistance de l'enseignant sur la nécessité de calculer le reste aurait conduit les élèves à écarter (provisoirement) l'idée selon laquelle la fonction MID$ modifie par effet de bord la chaîne argument. Cependant si l'on examine attentivement le compte rendu de la résolution du problème **FILE** (annexe **10**, séance du 19 avril), on se rend compte que spontanément, les élèves font des propositions tendant à l'élimination du premier caractère: LISTE$=LISTE$ - 1, puis LISTE$=LISTE$ - "1" (**Karine-Natacha**); LET LISTE$=LISTE$ - MID$(LISTE$,1,1) (**Arnaud-Jérôme**); les circonstances ne suffisent donc pas à expliquer cette différence de comportement.
- les objets intuitifs et la situation sur lesquels porte le problème sont différents: dans un cas, on calcule le premier caractère, et on l'affiche; les élèves savent par expérience que le fait d'afficher la valeur d'une variable ne supprime pas cette valeur; d'où la nécessiter de l'"ôter" de la liste; dans le second cas, on calcule le premier caractère, et on le concatène à droite du mot; les élèves peuvent alors faire l'analyse que si cette opération conservait ce caractère à la première place, il serait

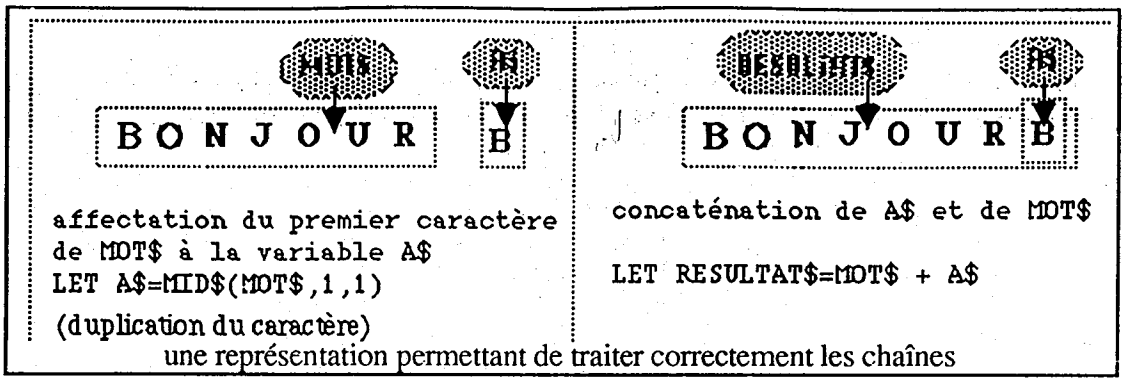
présent à la fois comme premier caractère et comme dernier; ceci est contradictoire avec leur *représentation des objets comme entités physiques*: un même objet ne saurait être en deux endroits différents. Explicitons cette représentation, par différence avec une représentation permettant de traiter correctement les chaînes et examinons ses conséquences sur la résolution par les élèves dans les deux situations:

- Dans une représentation permettant de traiter correctement les chaînes, les objets manipulés sont des valeurs qui sont copiées ou substituées; ainsi l'instruction `LET A$=MID$(LISTE$,1,1)` duplique le premier caractère de `LISTE$` et l'affecte, en le substituant éventuellement à une valeur déjà existante à la variable `A$`; on peut dire que `A$` a pour valeur le premier caractère de `LISTE$`, ou de façon plus rapide "est le premier caractère de `LISTE$`", mais ce n'est vrai que de façon provisoire: si l'on change la valeur de `LISTE$` par une affectation, la valeur de `A$` ne s'en trouvera pas modifiée.
- Dans la représentation des objets comme entités physiques (non duplicables), l'instruction `LET A$=MID$(LISTE$,1,1)` permet que la variable `A$` repère le premier caractère de `LISTE$`, mais ce caractère n'est pas compris comme une valeur dupliquée; l'identificateur de variable `A$` est compris comme une "*étiquette*" désignant ce premier caractère, qui lui, reste physiquement présent dans la chaîne `LISTE$`.[124]
- Dans la première situation (gestion d'une file d'attente), on affiche ce premier caractère (instruction `PRINT A$`); il n'est pas considère comme éliminé par cet affichage; c'est pourquoi les élèves reconnaissent qu'il convient de l'"*ôter*" de `LISTE$` pour mettre à jour la file..
- Dans la seconde situation (celle du problème 1 de l'épreuve sur papier), l'instruction `LET A$=MID$(CHAINE$,1,1)` permet de même que la variable `A$` repère le premier caractère de `CHAINE$`; si on le place "à la fin", la représentation des objets comme entités physiques conduit à le considérer comme physiquement déplacé; en effet `A$` n'est pas, dans cette représentation une simple copie du premier caractère, **mais bien le caractère lui-même**; si donc ce caractère est physiquement déplacé, il ne convient pas de l'"ôter" de la première place, ce qui explique la réponse de **Karine** et **Jérôme** .



la représentation des objets comme entités physiques (non duplicables)

[124] Il s'agit bien dans les deux cas de *représentation mentale* de la façon dont les objets sont traités dans le dispositif : la réalité (c'est-à-dire la façon dont les chaînes sont effectivement traitées comme partie de la mémoire peut différer notablement de ce que nous appelons *une représentation permettant de traiter correctement les chaînes*. En particulier, comme nous l'avons vu au chapitre 5, dans certaines réalisations, des identificateurs différents pourraient "partager" des chaînes physiques en mémoire.



une représentation permettant de traiter correctement les chaînes

Cette analyse nous permet d'interpréter de façon plus complète l'écriture de **Karine** présentée ci-dessus: dans la résolution de FILE, ayant affecté à `A$` le premier caractère de `LISTE$`, les élèves ont à "ôter" de `LISTE$` ce premier caractère; **Karine** propose de façon insistante (mais sans l'expliquer), l'écriture `B$ = MID$(A$, L-1,1)`. Nous avons dit ci-dessus que l'élève considère comme premier argument la chaîne qui lui semble définir le problème à résoudre, ici le premier caractère, qui est à "ôter", la chaîne `LISTE$` (celle dont le résultat est une sous-chaîne), étant pour elle implicite. En fait, `A$` reste bien pour l'élève le premier caractère de `LISTE$`, en tant qu'objet non duplicable, et non une copie; ainsi, `MID$(A$, L-1,1)` code bien l'action de retirer le premier caractère de `LISTE$`, sans qu'il soit nécessaire de préciser cette chaîne dans les arguments.

Nous retiendrons cette représentation des objets, comme une explication des comportements de **Karine** et **Jérôme** face aux deux situations, et nous en tirons deux conclusions:

- deux situations peuvent être proches pour certains élèves (ici **Natacha**) disposant d'une représentation adéquate des objets; pour d'autres (**Karine**, **Jérôme**), l'une des situations ne remet pas en cause leur représentation erronée des objets, alors que la deuxième entre en contradiction avec cette représentation , ce qui entraîne une solution erronée.
- l'hypothèse selon laquelle, certains élèves interprètent la fonction sous-chaîne comme modifiant par effet de bord la chaîne argument, demande à être précisée; en effet nous constatons ici que cette interprétation de la fonction `MID$` n'apparaît pas dans toutes les situations: c'est la *représentation des objets comme entités physiques non duplicables* qui est en cause, et qui détermine les situations où apparaît cette interprétation de la fonction `MID$`.

7.4. Les difficultés liées au calcul des arguments numériques (*D.Ord.Card.*)

7.4.1. La translation (ajout d'un cardinal à un ordinal)

Les problèmes **ALPHA2** et **PIANO1** sont conçus pour que les élèves effectuent une telle construction, l'ordinal étant une variable, et le cardinal une constante; dans **ALPHA2** , les élèves tentent d'abord des formulations naïves (`MID$(ALPHABET$,N,MINUSCULES)`), puis construisent cette translation. Dans **PIANO1**, la translation est construite sans difficulté.

7.4.2. Les difficultés concernant les ordinaux et la fonction longueur

Nous avons vu plus haut qu'un ordinal peut être, pour des élèves résolvant un problème sur les chaînes, confondu avec le caractère dont il est la position; nous avons rappelé en particulier les difficultés qui résultent du **double statut d'ordinal** (position de la dernière lettre de la chaîne) **et cardinal** (nombre d'éléments) de la fonction `longueur`; les problèmes de la série **TESTC** ont été construits en fonction de ces difficulté: dans **TESTC1**, l'ordinal résultat de la fonction longueur est reconnu comme la position du dernier caractère; on retrouve cependant chez **Karine** l'idée que `L` (résultat de la fonction longueur) représente "tout le mot" .

La recherche de la position du caractère médian dans **TESTC2** conduit tous les élèves à la formulation spontanée `L/2`, alors que `L`, la longueur de la chaîne considérée, est impair: "`L` c'est toute la chaîne, donc `L/2` c'est la moitié, donc le

milieu"; L/2 est, dans les représentations des élèves à la fois une position (le milieu), une partie de la chaîne, et un nombre de caractères (la moitié); l'indifférenciation entre une notion cardinale (la moitié) et une notion ordinale (le milieu) s'observe dans des représentations où les entités numériques ne sont pas distinguées des objets[125]. Au chapitre 6, nous avions observé ces représentations dans le cas de L lui même: en tant que position du dernier caractère, c'est un ordinal; en tant que nombre de caractères, c'est un cardinal, mais c'est aussi "toute la chaîne". Dans la résolution de **TESTC2**, la formulation spontanée du milieu (L/2), et les difficultés de résolution qui en découlent conduisent les élèves à un déséquilibre: recevant en retour du dispositif une erreur d'exécution, les élèves doivent reconnaître que L/2 a une valeur décimale non entière, donc sont conduits à *reconnaître* L *comme entité numérique* , distincte d'une partie de la chaîne; admettant que la position du milieu doit être un entier, ils reconnaissent également à cette position son statut numérique: en ce sens, ils progressent vers une meilleure représentations de la fonction longueur et des ordinaux.

TESTC2 conduit donc bien les élèves à des différenciations dans leurs représentations. Leur donne t'il des outils permettant de mieux dominer les calculs sur les ordinaux? Sans doute non, puisque les élèves se contentent d'une procédure d'ajustement empirique; il aurait été sans doute intéressant de proposer ensuite des calculs sur des "milieux" plus généraux, de façon à progresser vers la construction d'une fonction qui aux ordinaux a et b fait correspondre l'ordinal (a + b) / 2 .

TESTC3 a été construit dans l'hypothèse que les élèves emploieraient spontanément une itération sur la moitié de la chaîne, avec un compteur pointant le caractère courant (voir variante1 dans la présentation de cet problème au chapitre précédant); dans cette hypothèse, les élèves auraient eu à calculer la position du caractère symétrique au caractère courant, ce qui aurait représenté un calcul complexe (pour eux) sur les ordinaux). En fait, seule la forme d'itération la plus récemment acquise (la forme REPETER <action> JUSQU'A <condition>) s'est trouvée spontanément disponible; cette forme rend possible l'utilisation de deux compteurs pointant respectivement un caractère et son symétrique; elle évite le calcul de la position du symétrique, et est plus générale; c'est pourquoi nous n'avons pas dissuadé les élèves de l'utiliser; mais les difficultés se sont trouvées reportées sur l'itération, les élèves traitant peu aisément un corps de boucle à plusieurs instructions. Un autre choix aurait pu consister à prévoir une question préliminaire, pour que les élèves soient confrontés au calcul de la position du caractère symétrique, par exemple sous l'énoncé suivant: *écrire un programme tel que l'utilisateur entre une position, l'ordinateur affiche le caractère ayant cette position, et le caractère symétrique.*

Utilisant pour l'itération la forme REPETER <action> JUSQU'A <condition> , **Karine-Natacha** ont rencontré cependant des difficultés propres aux ordinaux pour la construction de la condition d'arrêt; lorsque la chaîne est de longueur impaire, l'égalité des deux compteurs constitue cette condition, et elle est facilement trouvée; lorsqu'elle est paire, **Natacha-Karine** proposent la condition X= L + 0.5 et ne comprennent pas que , X et L étant entiers, la condition ne saurait être vérifiée;; elles restent influencées par l'idée que les compteurs se retrouvent "au milieu de la chaîne" donc entre deux caractères, quand la chaîne est de longueur paire, d'où le 0.5, ce qui est à mettre en relation avec les difficultés rencontrées dans **TESTC2** avec la notion de milieu; la production d'une condition d'arrêt correcte (X = L -1) constitue pour ces élèves un réel progrès.

Jérôme résout **TESTC3** seulement dans le cas d'une chaîne de longueur paire, et ne rencontre pas la même difficulté que **Karine-Natacha** dans l'écriture de la condition d'arrêt; c'est regrettable, car, comme nous le soulignons ci-dessus, cette difficulté s'est révélée productive dans le cas de cet autre groupe; sa recherche témoigne néanmoins d'une confusion entre caractère et position; cette confusion se traduit par les affectations LET A$= A-X et LET B$ = A - X; A ayant reçu la

---

[125]La confusion entre milieu et moitié s'observe couramment en géométrie, mais elle ne porte pas nécessairement à conséquence, car les points sont bien distingués des quantités numériques ; d'autre part l'origine O d'un repère sur une droite ayant comme abscisse zéro, et un point M ayant comme abscisse m, l'abscisse du milieu de OM est effectivement m/2.

valeur L/2, repère une position médiane dans la chaîne, et X est un compteur d'itération; l'erreur de type que constituent ces affectations est détectée par le compilateur, ce qui permet à l'élève de corriger en:

```
LET A= A-X
LET B= B+X
LET A$= MID$(CHAINE$,A,1)
LET B$=MID$(CHAINE$,B,1)
```

A l'épreuve sur papier, seule la réponse de **Natacha** comporte une écriture erronée, dont la confusion entre un ordinal et la position qu'il occupe dans une chaîne est partie prenante; nous avons analysé ci-dessus cette écriture en lien avec la difficulté concernant les types: `LETTRE1$ = MID$(ALPHABET$,LETTRE$ + 1,1)`. Il y a en effet une erreur de type dans l'écriture `LETTRE$ + 1` , mais plus qu'un simple défaut de syntaxe, cette erreur révèle la persistance d'une confusion entre ordinal et caractère. Comme nous l'avons mis en évidence au chapitre 7, ce type de conduite résulte de la conjonction d'une représentation des objets ne différenciant pas les ordinaux des caractères qu'ils repèrent, et d'une représentation de l'accès aux parties d'une chaîne (accès direct ou associatif).

A titre de bilan nous retiendrons que, au cours de la résolution des problèmes, les élèves ont rencontré des difficultés liées au calcul des ordinaux et que leurs représentations ont évolué; des représentations antérieures peuvent néanmoins réapparaître, notamment quand elles se trouvent en conjonction avec une représentation du traitement.

7.5. l'interaction entre représentations des objets, et représentation des traitements; les problèmes de "recherche translation" et de comptage d'occurrences

Le chapitre **7** de cette thèse est consacré à l'étude des conduites résultant de l'interaction entre représentations des objets, et représentation des traitements. Nous avons mis en évidence, dans deux problèmes de "recherche-translation", portant sur les chaînes de caractères, en tant qu'objets, et sur une forme simple d'itération, en tant que traitement, que les conduites des élèves s'analysent comme des points d'équilibre, conjonction d'un plan relatif aux données et d'un plan relatif au traitement. Nous avons vu, ponctuellement, au cours d'un entretien, qu'une remise en cause d'un plan relatif aux données peut conduire à une réequilibration dans la conception du traitement. Les problèmes de type *CONSTRUCTION ITERATION*, des séries **ALPHA**, **PIANO** et **TESTCHA**, ont été proposés aux élèves, dans le but d'étudier dans la durée cette interaction des représentations.

7.5.1 Les acquisitions recherchées, et les problèmes proposés

Dans ces problèmes, la tâche consiste à construire le parcours itératif d'une chaîne dans le but d'en extraire une information; nous avons voulu ainsi, compte-tenu des difficultés repérées dans ce domaine, limiter les ambitions à une forme simple d'itération comportant un parcours linéaire et un seul point de sortie; nous avons voulu également, puisque c'est le sujet de notre recherche, que les représentations des élèves concernant la structure de chaîne et la structure ordinale qui lui est associée soient sollicitées lors de la construction de l'itération. Cela conduit à deux classes de problèmes:

- Les problèmes de "**recherche-translation**", déjà présentés au chapitre **7**: la donnée est un caractère (ou plus généralement une sous-chaîne) de la chaîne à parcourir, et le résultat est un caractère image du caractère donné dans une translation. Appartiennent à cette classe:
- **ALPHA4** (la chaîne est formée des 26 lettres de l'alphabet majuscules, suivies des 26 lettres de l'alphabet minuscules; on demande d'entrer une majuscule, et d'afficher la minuscule correspondante) ,
- **PIANO1** (la chaîne ayant pour valeur "`DO C RE D MI F FA G SOLH LA A SI B FIN "` , entrer une chaîne de deux ou trois caractères (`DO, RE ... SI`) et calculer la lettre correspondante (`C ,D , ..., A, B`)),
- **CRYPTAGE** (problème 3 de l'épreuve sur papier), (la chaîne étant

"azertyuiopqsdfghjklmwxcvbna" une lettre de l'alphabet étant donnée, le résultat est la lettre qui la suit dans la chaîne) .

Dans ce type d'énoncé, les élèves peuvent adopter deux formes d'itération:

- première forme: l'utilisation d'une itération sur toute la chaîne, le corps de boucle étant une alternative: N étant la valeur de la translation à effectuer, et L la longueur de la partie de la chaîne où s'effectue la recherche, cette forme s'écrit:

```
POUR I←1 JUSQU'A L
  SI DONNEE = sous-chaîne(CHAINE,I,1)
        ALORS RESULTAT ← sous-chaîne(CHAINE,I + N,1) IS
FINPOUR
```

Dans une variante, le corps de l'itération peut être séparé en deux instructions:

```
LETTRECOURANTE ← sous-chaîne(CHAINE,I,1)
SI DONNEE = LETTRECOURANTE
        ALORS RESULTAT ← sous-chaîne(CHAINE,I + N,1) IS
```

Cette forme a été employée spontanément dans la classe où nous avons fait des observations en 86-87, et dans la classe où nous avons procédé à une pré-expérimentation en 87-88; ces classes utilisaient une version de BASIC où la seule forme d'itération structurée disponible était de ce type, les autres formes classiques (TANT QUE , REPETER JUSQU'A devant être construites à l'aide d'instructions de branchements GOTO). Elle peut être considérée comme lourde et peu efficace: en effet, le parcours de la chaîne se poursuit, même après que le résultat ait été calculé. Des difficultés de l'itération repérées comme importantes chez les élèves n'apparaissent pas dans cette forme: initialisation des variables itératives, construction de la condition d'arrêt; en fait, plus qu'une itération, les élèves la considèrent comme le résumé de L alternatives.

- seconde forme: utilisation d'une forme structurée d'itération, avec condition d'arrêt, par exemple REPETER...JUSQU'A:

```
I ← 0
REPETER
        I ← I + 1
JUSQU'A DONNEE = sous-chaîne(CHAINE,I,1)

RESULTAT ← sous-chaîne(CHAINE,I + N,1)
```

De même que dans la forme précédente, on a une variante en employant l'affectation dans le corps d'itération:

```
I ← 0
REPETER
        I ← I + 1
        LETTRECOURANTE ← sous-chaîne(CHAINE,I,1)
JUSQU'A DONNEE = LETTRECOURANTE

RESULTAT ← sous-chaîne(CHAINE,I + N,1)
```

Cette forme est plus générale et plus efficace, mais elle recèle une accumulation de difficultés: initialisation, séquentialité dans le corps de boucle (dans la variante employant l'affectation), construction de la condition d'arrêt et articulation avec le corps de boucle; d'autre part, il convient de séparer le programme en deux parties: la recherche de la position du caractère donné, et le calcul du résultat; la variable I qui était un simple paramètre de boucle au statut pouvant rester assez flou dans la première forme, devient ici un résultat intermédiaire au rôle décisif.

- les problèmes de **comptage d'occurrences**: la donnée est la chaîne à parcourir, et le résultat un cardinal résultant d'un comptage d'occurrences dans la chaîne: un problème classique consiste par exemple à demander le calcul du nombre d'occurrences d'un caractère donné dans une chaîne donnée; nous avons voulu que le calcul sur les ordinaux soit davantage sollicité que dans cet problème; c'est pourquoi nous avons proposé le problème **TESTC3**, où le comptage met en jeu deux positions évoluant dans la chaîne (il s'agit dans cet problème de compter le nombre de paires symétriques dont les caractères sont différents; si le résultat est zéro, la chaîne est un palindrome). Nous avons montré dans la présentation du problème au chapitre précédent, que pour cette classe de problèmes également, deux formes d'itération sont possibles: POUR ...FINPOUR et REPETER..JUSQU'A...; la seconde forme est plus générale, mais elle contient davantage de difficultés liées à l'itération.

7.5.2 Il a été possible d'utiliser seulement une forme d'itération avec les élèves

Nous avons vu que dans les deux classes de problèmes, le choix se fait en principe entre une forme lourde, peu généralisable (POUR... FINPOUR), et une forme plus générale, mais où les difficultés liées à l'itération sont plus présentes. Nous faisions l'hypothèse que les élèves produiraient d'abord la première forme, de façon à obtenir une réussite au moins pour les premiers problèmes et à travers cette réussite atteindraient un début de compétence dans le domaine de l'itération, puis qu'en posant des problèmes plus généraux, nous pourrions les amener à envisager l'autre forme. Nous avions pensé à ce schéma en proposant la série **PIANO** après la série **ALPHA**, et **TESTC4** après **TESTC3**.

Mais ce schéma s'est révélé inadapté; il suppose en effet, que les élèves maîtrisent suffisamment l'itération pour que les deux formes soient disponibles simultanément, et que les rapports entre les deux formes soient suffisamment clairs pour eux. Il est apparu rapidement que ce n'est pas le cas des élèves observés: seule la forme d'itération REPETER...JUSQU'A..., la plus récemment utilisée s'est révélée disponible[126]; c'est pourquoi nous n'avons pas voulu orienter les élèves vers l'autre forme, qui aurait constitué un retour en arrière coûteux en temps. En fait il aurait été préférable de considérer d'emblée que pour ces élèves, et à ce niveau, une seule forme d'itération est envisageable; dans cette hypothèse, la forme POUR...FINPOUR n'est pas la meilleure, même si elle peut paraître plus facile à mettre en oeuvre; étant trop particulière, elle pourrait gêner le passage ultérieur à une itération plus générale. Nous considérons, avec le recul donné par l'expérimentation dans une classe disposant de la forme REPETER...JUSQU'A, que l'utilisation préférentielle de la forme POUR...FINPOUR dans la classe où nous avons mené la pré-expérimentation résulte en fait simplement des contingences de la forme de BASIC utilisée.

7.5.3 Les difficultés des élèves concernant l'itération observées au cours de la résolution des problèmes

- Résolution du problème **ALPHA3** (séance 2) (il s'agit de donner comme résultat la position d'une lettre donnée dans l'alphabet)
  Karine écrit assez facilement un programme exécutable mais les difficultés qu'elle rencontre au problème suivant (**ALPHA4**) montrent que cette élève n'a pas conscience de réaliser une itération: les avertisseurs d'itération et l'instruction d'incrémentation du compteur de boucle ont été placés pour répondre à la demande de l'enseignant, mais pour l'élève, seule la condition LETTRE$=MID$(ALPHABET$,X,1) a une importance. Nous avons vu en effet que cette élève confond les notions de condition et d'affectation et les assimile à une forme "déclarative" qui aurait ici pour effet de donner à X la valeur du rang de LETTRE$ dans ALPHABET$.
  La même confusion est présente chez **Arnaud-Jérôme** , mais leur choix de réaliser une affectation dans le corps de boucle les conduit aux difficultés classiquement repérées dès que le corps de boucle comprend plus d'une instruction; résolvant ces difficultés, de façon laborieuse, ils prennent davantage conscience de l'itération qu'ils construisent, et de l'évolution des variables.
- La résolution de **ALPHA4** (séance 2): (ALPHABET$ a cette fois comme valeur une chaîne formée des 26 lettres de l'alphabet majuscules, suivies des 26 lettres de l'alphabet minuscules; on demande d'entrer une majuscule, et d'afficher la minuscule correspondante). Ce problème est du type "**recherche-translation**"; davantage que **ALPHA3**, il sollicite la structure "chaîne de caractères", puisqu'on a un caractère en entrée, et un caractère en sortie; ainsi, l'intervention d'un résultat intermédiaire numérique peut ne pas apparaître comme nécessaire, le traitement par

---

[126] En disant que cette forme est "disponible", nous ne voulons pas signifier que cette forme d'itération est acquise, ou même comprise par les élèves, mais simplement qu'à l'idée de boucle, ou de résolution d'un problème itératif, les élèves associent la forme syntaxique DO..LOOP UNTIL... qui, dans la variété de BASIC utilisée est la traduction de la forme REPETER...JUSQU'A... ; la forme syntaxique FOR...NEXT, traduction de POUR...FINPOUR, était quant à elle, totalement oubliée par les élèves.

un opérateur humain n'y ayant pas recours; cependant, en construisant la série **ALPHA**, nous avions pensé préparer les élèves à cette difficulté par la résolution de **ALPHA2** (où ils aperçoivent la nécessité d'une translation pour obtenir une minuscule, connaissant le rang de la majuscule) et **ALPHA3** (où ils construisent une itération pour obtenir le rang d'une majuscule); il nous avais semblé qu'il serait aisé de réutiliser ces deux solutions pour construire une solution de ALPHA4.
La réalité est bien différente: **Karine** pense à réutiliser la translation, mais ne comprend pas que **ALPHA3** peut lui permettre d'obtenir le rang de la majuscule à partir duquel opérer cette translation; au lieu de cela, elle propose des formulations "directes" telles que `LET LETTRE$=X`, que nous avons analysé en lien avec les difficultés de cette élève concernant l'affectation. Ensuite, elle accepte de construire une itération, mais cette itération est manifestement sans signification pour elle puisque la condition d'arrêt se confond avec l'affectation permettant d'obtenir le résultat; l'élève ne parvient pas à surmonter cette difficulté dans le temps imparti.
Jérôme -Arnaud disposent de peu de temps pour cette résolution; mais ils se montrent eux-aussi incapables d'articuler leurs solutions à **ALPHA2** et **ALPHA3** pour résoudre **ALPHA4.**

La résolution de **ALPHA4** se termine donc par un échec pour les 3 élèves. Les difficultés rencontrées dans cet problème surprennent si l'on s'attendait à ce que les élèves reconnaissent dans **ALPHA4** la juxtaposition de **ALPHA2** et **ALPHA3**; tentons de les analyser.

Pour l'élève, la nécessité d'une itération pour obtenir la position de la lettre donnée, peut ne pas apparaître et il y a confusion entre cette position et le caractère lui-même; en effet dans **ALPHA3**, pour obtenir le rang d'une lettre dans l'alphabet, il est assez naturel de COMPTER, puisqu'on veut un résultat numérique; par contre, dans **ALPHA4**, pour un opérateur humain, il n'y a pas besoin de chercher itérativement le rang de la lettre donnée; le repérage de la lettre donnée peut se faire de façon directe, par lecture globale de la chaîne, et la translation qui conduit à la lettre résultat se fait de façon relative (en comptant à partir de la position trouvée pour la lettre donnée). Les élèves éprouvent des difficultés à construire une itération dont ils ne perçoivent pas la nécessité, et à calculer sur une donnée dont le type ne leur apparaît pas clairement. Comme au chapitre 7, ces difficultés apparaissent comme conjonction d'une représentation des objets et d'une représentation des traitements

Cependant, ayant accepté l'idée d'une itération, ils rencontrent d'autres difficultés qui tiennent à la séparation du programme en deux modules, le résultat du premier module étant la donnée du second. **Karine,** par exemple, a écrit le second module (calcul du caractère résultat, le rang étant supposé trouvé). Pour terminer le programme, il lui suffirait de recopier la solution de **APHA3**, qui constitue précisément le calcul du rang. Au lieu de cela, elle s'obstine à "fusionner" les deux modules, *l'affectation* du second module prenant la place de la *condition de sortie* du premier! Nous trouvons ici une forme assez particulière des conduites que nous mis en évidence au chapitre 7: une représentation des objets différenciant mal l'affectation et les conditions interagit avec une compréhension insuffisante des éléments de l'itération, créant un point d'équilibre empêchant la coordination (pourtant simple) des deux modules.

Cette conduite est liée au choix de la forme d'itération. Elle ne se serait pas rencontrée si les élèves avaient employé la structure `POUR...FINPOUR`, où le calcul du résultat se fait à l'intérieur du corps de boucle. A la suite de la pré-expérimentation, nous avions fait l'hypothèse que les élèves construiraient spontanément une structure itérative du type `POUR..FINPOUR`, comme résumé de 26 alternatives; comme nous l'avons expliqué plus haut, il nous est apparu, à partir de la 2ème séance qu'il ne serait pas possible d'employer avec les élèves deux formes d'itération distinctes,ces élèves ne dominant pas suffisamment ces formes pour faire un choix et passer facilement de l'une à l'autre; nous avons dû par conséquent nous limiter à l'emploi de la structure `REPETER...JUSQU'A...` qui constituait la connaissance la plus récente, mais aussi la forme la plus générale, parmi les deux que nous avions envisagé en préparant les problèmes. Ce défaut de l'analyse préalable nous a conduit à introduire sans doute trop tôt (2ème séance) ces difficultés: les

représentations des objets étant encore peu différenciées, il n'était guère possible de les faire évoluer parallèlement aux représentations des traitements.

- résolution de **PIANO1** (séance 3): ce problème est du même type que **ALPHA4** ("recherche-translation" dans une chaîne); le problème diffère dans la mesure où la donnée n'est pas un caractère, mais une chaîne de longueur variable. La lecture du compte rendu ne fait pas apparaître cette différence comme conduisant à une difficulté dominante parmi celles que rencontrent les élèves observés. Les difficultés apparues dans **ALPHA4** sont par contre très présentes:
    - mise en évidence de la position du premier caractère de la note donnée, comme résultat intermédiaire, et séparation en deux modules,
    - statut exact, et mode de calcul de cette position, et donc écriture de la condition d'arrêt.

La séparation en deux modules paraît cependant mieux intégrée, la fixation de **Karine** sur la "fusion des modules" observée dans **ALPHA4,** n'apparaît pas ici. Les difficultés se concentrent ainsi sur la condition de sortie, qui fait intervenir à la fois la représentation du traitement et la représentation des données: dans les deux groupes, cette condition est écrite d'abord de façon naïve, ensuite le statut flou de l'ordinal intervient dans les écritures. Il semble qu'il y ait un certain progrès à la fois dans la représentations du traitement (l'itération est mieux comprise, même si la condition de sortie reste un point difficile), et dans la représentation des objets (confusion condition-affectation moins prégnante, début de clarification du statut de l'ordinal), mais le schéma selon lequel s'opère ce progrès n'apparaît pas clairement: nous n'observons pas le schéma déséquilibre dans les objets, réequilibration dans le traitement mis en évidence au chapitre 7.

- Résolution de **TESTC3** (séances 4 et 5); ce problème est du type "comptage d'occurrences"; les élèves ne sont pas tentés de donner une solution non-itérative; en effet, la nécessité d'une itération s'impose comme proche d'une résolution par un opérateur humain. Le compte rendu fait apparaître que les difficultés propres à l'itération dominent ici. Parmi celles-ci, l'initialisation des variables, et la prise en compte de la séquentialité dans le corps de boucle, apparaissent relativement indépendantes des représentations des objets. Par contre, dans ce contexte itératif, l'emploi de la fonction `MID$` (sous-chaîne) est compris d'abord comme pouvant faire évoluer les variables (ici des index sur des caractères de la chaîne) par effet de bord; il y a donc interaction entre la difficulté à concevoir des variables itératives, et la compréhension naïve de la fonction `MID$`. La construction de la condition d'arrêt donne lieu dans les deux groupes à des emplois erronés de la fonction `LONGUEUR`; **Jérôme -Arnaud** résolvent seulement dans le cas d'une chaîne de longueur impaire, et le parcours de la chaîne à partir du milieu leur permet d'obtenir plus facilement cette condition; dans le cas de **Karine-Natacha**, l'écriture d'une condition valable quelque soit la parité de la longueur de la chaîne, se révèle un problème non-immédiat, mais à leur portée; leur réussite constitue un progrès dans l'acquisition de la structure. Nous retiendrons donc que, parmi les difficultés propres à l'itération, dans le cas d'un problème de comptage d'occurrences, ce sont essentiellement l'évolution des paramètres de boucle, et la construction de la condition d'arrêt qui interagissent avec les représentations des objets.
- résolution de **CRYPTAGE** (problème 3 de l'épreuve sur papier) il s'agit d'un problème de "recherche-translation". Un des élèves observés (**Jérôme**) présente une solution pleinement satisfaisante: une affectation à l'intérieur de la boucle donne une valeur au caractère courant, ce qui porte à deux le nombre d'instructions du corps de boucle; malgré cette difficulté supplémentaire, l'initialisation, la séquentialité dans le corps de boucle, la condition d'arrêt, et le calcul du résultat sont correctement articulés. Ce succès montre que la résolution de problèmes de "recherche-translation" est un objectif raisonnable, y compris pour des élèves présentant des difficultés d'acquisition. Deux autres élèves échouent; **Karine** sépare le problème en deux modules élémentaires, mais le premier module (qui doit calculer la position du caractère donné) traduit la persistance d'une compréhension "incantatoire" de l'affectation, fréquemment rencontrée antérieurement chez cette élève (voir 7.2 .2); la présence d'une instruction `LET X=X+1`, qui ne prend son sens que dans le corps d'une itération, témoigne que l'élève a pu envisager une itération,

mais sa compréhension de l'affectation ne lui permet pas de différencier condition d'arrêt et affectation; la solution de **Natacha** est celle que, au chapitre **7**, nous avons caractérisée sous l'appellation *"accès direct"*

7.6. l'intervention de structures alternatives:

La structure alternative (si <condition> alors<action1> sinon <action2>) peut présenter des difficultés dans le cas où la condition est complexe, c'est-à-dire, fait intervenir plusieurs variables. Il peut se poser également la question de l'interprétation de la structure alternative dans le cas où <action1> et/ou <action2> sont des actions complexes, faisant évoluer des variables. La structure d'alternatives en chaîne constitue une instance particulière de ce cas, celle où <action1> et/ou <action2> sont elle-même des alternatives, et nous avons vu au chapitre **2**, avec la présentation de [GREEN 77] la complexité des modes de fonctionnement mental nécessaire à l'interprétation de cette structure. Bien que ne constituant pas l'objectif central de l'observation rapportée dans ce chapitre, la structure alternative intervient, sous les deux aspects dans plusieurs problèmes de la progression, et nous faisons donc le point sur les difficultés rencontrées.

7.6.1 Le cas d'une condition complexe:

L'énoncé **TESTC2** a été construit pour envisager ce cas: il s'agit de comparer trois caractères d'une chaîne (le premier, le dernier, et celui du milieu). Dans les deux groupes d'élèves observés, les caractères sont respectivement affectés à A$, B$ et C$ et la condition est rédigée sous la forme A$=B$=C$. Confronté à une erreur de compilation, le groupe **Karine-Natacha** sépare en deux alternatives:

```
IF A$= B$ THEN PRINT "OUI" ELSE PRINT "NON"
IF B$= C$ THEN PRINT "OUI" ELSE PRINT "NON
```

Mais cette structure conduit à 4 réponses distinctes (OUI OUI, OUI NON, NON OUI, NON NON), et non aux deux attendues (OUI, NON); les élèves reviennent à une seule alternative, en utilisant le connecteur logique AND, après s'être assuré auprès de l'enseignant que ce connecteur peut s'employer ici:

```
IF A$= B$ AND B$=C$ THEN PRINT "OUI" ELSE PRINT "NON"
```

L'autre groupe passe directement à cette forme. Il n'y a donc dans ce processus de résolution, rien qui apparaisse comme spécifique des chaînes de caractères; le même processus (expression spontanée, puis utilisation d'un connecteur logique s'observe dans le domaine numérique); il en aurait peut-être été autrement si les élèves n'avait pas utilisé l'affectation, les conditions étant dans ce cas d'écriture très lourde, et pouvant mener à des confusions.

7.6.2 le cas où <action1> et/ou <action2> sont des actions complexes

Cette question se pose dans deux problèmes, où les élèves, après avoir proposé une résolution qui les satisfont, ont leur attention attirée par un enseignant sur un cas particulier qu'il n'ont pas envisagé: dans **INSEE3**, après avoir écrit un programme permettant de calculer l'age d'une personne connaissant son numéro INSEE, les élèves sont confrontés au cas où le titulaire du numéro est né avant 1900; les deux groupes placent d'emblée l'alternative en *fin de programme*, se mettant alors en situation d'apporter une *correction* au résultat calculé: l'alternative prend alors la forme SI <cas particulier> ALORS <correction> (il aurait été possible de placer cette alternative plutôt au moment du calcul de l'année de naissance); la partie condition ne leur pose pas de difficulté particulière; par contre la rédaction de la partie <correction> est difficile à obtenir; cette partie prend en effet la forme d'une affectation à un identificateur de variable, d'une expression comportant cette variable; cette forme se rencontre également pour l'évolution des variables itératives (compteur, variables cumulatives) dans une itération, et on peut remarquer que les élèves ne l'emploient pas volontiers.

La question se présente également dans le premier module de **FILE**, où après avoir traité l'évolution de la file dans le cas ou un patient est reçu par le médecin, les élèves doivent envisager le cas particulier où la salle d'attente serait vide. Ce cas est d'autant plus délicat qu'il s'insère dans un module qui est lui même une branche d'une

alternative; les deux groupes insèrent l'alternative en fin de module sous la forme `IF LISTE$="" THEN PRINT "Plus de patients"`; ceci conduit à une erreur à l'exécution, puisque la première partie du module suppose que `LISTE$` soit non vide; une écriture correcte suppose de concevoir la première partie du module comme un bloc placé dans la partie <action2> qui suivrait un `ELSE` dans l'alternative; on constate à la lecture du compte rendu que les élèves ne sont pas capables de surmonter seuls une telle difficulté d'articulation.

Il n'y a pas non plus dans cette question de difficulté semblant spécifique aux chaînes de caractères; les progrès des élèves apparaissent d'ailleurs comme très lents dans ce domaine; la question des cas particulier a été posée parce qu'elle se présentait dans les situations proposées en problème; mais les séries n'avaient pas particulièrement pour but de faire progresser les élèves dans ce domaine.

8. La situation d'apprentissage

Nous avons indiqué au paragraphe 3 nos choix pour la mise en place des 9 séances: bien qu'éloignées du schéma indiqué par le programme officiel de l'option, les conditions dans lesquelles les élèves sont placés sont celles que l'on trouve habituellement dans cette classe, et sans doute dans d'autres classes de l'option informatique, au niveau de l'alphabétisation. Nous pouvons à ce stade difficilement faire des hypothèses, faire varier ces conditions et mesurer l'effet de ces variations. Nous relevons cependant dans cette situation de résolution un certains nombre de traits qui paraissent pertinents: concernant l'élève, ce sont les utilisations des formulations en langage naturel, et des réponses du milieu (c'est-à-dire du problème et du dispositif); concernant l'enseignant, ce sont les moyens dont il dispose pour orienter son intervention.

8.1 L'utilisation par les élèves des formulations en langage naturel

Les comptes rendus confirment les difficultés rencontrées par les élèves lorsqu'ils tentent d'utiliser des formulations en langage naturel au cours de la résolution, que ces formulations portent sur leur compréhension du problème, sur leurs intentions, ou sur leur interprétation des réponses du dispositif; d'une part, leur expression personnelle est limitée, d'autre part, les explications verbales plus ou moins métaphoriques de l'enseignant, les analogies, ne sont pas comprises, ou sont prises au pied de la lettre. Ainsi, certaines phrases de l'énoncé, certaines indications de l'enseignant sont traduites mot à mot par les élèves en BASIC. De même, l'échange à l'intérieur du groupe est limité par ces difficultés d'expression; un élève ne peut expliquer ses intentions à l'autre membre du groupe: suivant les phases, un des élèves du groupe propose des expressions BASIC correspondant à ses intentions (en fait, il ne les exprime pas oralement, il les écrit dans l'éditeur du langage de programmation); quand ces expressions conduisent à une erreur, l'autre membre du groupe propose alors ses propres expressions. Les conceptions diffusent peu à l'intérieur du groupe: ainsi **Natacha** et **Karine** présentent tout au long de l'observation des conceptions assez distinctes; ces conceptions évoluent plutôt dans un sens favorable, mais elles semblent peu s'influencer.

Nous ne pensons pas que des élèves parvenus en classe de Seconde de Lycée soient dépourvus de moyens d'expression, même si les élèves observés ici constituent peut-être des cas limites. Les élèves observés aux chapitres **6** et **7** semblaient disposer de davantage de moyens d'expression; le dépouillement des séances enregistrées montre cependant que leur expression reste superficielle, souvent limitée à une paraphrase de l'énoncé, et ceci d'autant plus que l'élève est face à une difficulté conceptuelle. Nous avons indiqué aux paragraphes 3 et 4 pour quelles raisons nous n'avons pas exigé des élèves une analyse en français courant avant le travail sur ordinateur. On sera peut-être tenté d'expliquer l'observation ci-dessus par cette absence d'exigence didactique; nous ne sommes pas de cet avis: en effet, l'échange en langage naturel est une nécessité de la situation même en l'absence d'une analyse préalable en français courant; nous constatons que les élèves tentent souvent de tels échanges, que ce soit entre eux ou en direction de l'enseignant. Ces tentatives ont

souvent peu de succès. Malgré cet insuccès, elles peuvent être importantes dans les progrès des élèves.

8.2 Le rôle de l'environnement du problème et des effets en retour du dispositif

Pour [BROUSSEAU 86], «*l'élève apprend en s'adaptant à un milieu qui est facteur de contradictions,de difficultés, de déséquilibres (et le) savoir, fruit de l'adaptation de l'élève, se manifeste par des réponses nouvelles qui sont la preuve de l'apprentissage*»; il est important que ce milieu soit perçu par l'élève comme «*sans intentions didactiques*», de façon qu'il puisse déterminer les réponses à apporter aux difficultés que lui crée le milieu, à partir de la logique de la situation elle-même, et non en fonction des intentions didactiques qu'il pourrait y deviner: si l'élève perçoit "ce qu'on veut lui faire dire", la situation perd son sens[127]. Cette notion d'un «*milieu a-didactique*» nous paraît pertinente pour caractériser l'intervention de deux facteurs inhérents au type de situation que nous analysons ici; ces deux facteurs sont d'une part de l'environnement du problème à résoudre, constitué d'objets du monde, d'autre part les effets en retour du dispositif.

8.2.1 Le rôle de l'environnement du problème; la production de valeurs de test par les élèves

La résolution d'un problème de programmation, même le plus infime, se place, dans la perspective d'une programmation "en vraie grandeur": celle où un programme doit fonctionner, non pour les besoins d'un apprentissage, mais pour réaliser une tâche ayant une utilité dans un environnement donné; c'est en cela que l'environnement du problème participe du «*milieu a-didactique*» dont parle [BROUSSEAU 86]. Ainsi, l'environnement impliqué par le problème doit permettre à l'élève de savoir si sa solution convient, si les structures qu'il a employées ou construites sont adaptées; il ne doit pas y être conduit par des exigences didactiques externes. Plus concrètement, on peut attendre de l'environnement du problème qu'il fournisse des valeurs de tests c'est-à-dire des couples [données, résultat] suffisamment génériques pour permettre un raisonnement garantissant la validité d'un programme, ou invalider un programme ne présentant pas le caractère de généralité attendu[128]. L'élaboration de valeurs de test est donc partie intégrante du problème (le programme officiel de l'option recommande d'ailleurs «*l'élaboration de jeux d'essais*»), et dans l'économie du système didactique, il est important que l'élève qui résout prenne en compte la généralité du problème, faute de quoi il peut ne pas rencontrer les difficultés, les déséquilibres auxquels on attend qu'il apporte des réponses nouvelles.

Or, la production de valeurs de tests suffisamment génériques, se heurte chez le programmeur à la difficulté de remettre en cause les limites de son analyse: la tendance spontanée est de produire des valeurs de tests conformes à sa propre analyse. Ainsi, les élèves n'ont pas de difficulté, dans les séries de problèmes, à déduire de l'énoncé le résultat qui serait obtenu avec les moyens propres à l'environnement, quand on leur propose des valeurs pour les données; de ce point de vue, l'environnement des problèmes joue son rôle dans le milieu a-didactique. Par contre, en ce qui concerne les valeurs des données à considérer, les élèves ont peu d'initiative, se contentant souvent des exemples de l'énoncé; l'enseignant est ainsi conduit à proposer lui-même les valeurs de tests, puisqu'il est important pour la progression de l'activité que les élèves prennent en compte la généralité du problème; ainsi par exemple, dans le calcul de l'age à partir du numéro INSEE (**INSEE2**, séance 7), les élèves se satisfont de données correspondant à une personne née après 1900; le système de tirage aléatoire de la donnée, prévu pour éviter aux élèves la tâche fastidieuse d'entrer à plusieurs reprises

---

[127]Par exemple, dans [LABORDE, BALACHEFF, MEIJAS 1985] le dispositif est un robot *simulé* par un des enseignants. Il nous semble qu'une telle situation pédagogique doit s'entourer d'un luxe de précautions pour que les élèves considèrent qu'une attitude donnée du simulateur, en réponse à un programme, est bien inhérente à la situation, et non à une intention didactique.

[128]Comme nous l'avons dit plus haut pour les essais en machine, les valeurs de test sont un support pour le raisonnement, et ne remplacent pas ce raisonnement.

un numéro à 13 chiffres impose plusieurs essais avant de produire un numéro correspondant à une personne née avant 1900. Les essais ne sont faits que sur l'intervention pressante de l'enseignant. Cette intervention de l'enseignant montre que le milieu ne joue pas pleinement son rôle ici, puisque ce milieu ne peut suffire à placer l'élève devant les contradictions attendues. Il y a donc un problème de «*dévolution*»: les élèves ne se sont pas appropriés le problème dans sa généralité.

8.2.2 Les effets en retour de l'ordinateur

Les effets en retour que les élèves obtiennent de l'ordinateur sont soit des messages d'erreur, soit des résultats d'exécution. Utilisant un langage qu'ils perçoivent comme non dédié aux apprentissages scolaires, les élèves peuvent comprendre ces effets en retour comme sans intentions didactiques, et donc comme des réponses du milieu.[129]

Examinons d'abord les messages d'erreur qu'ils obtiennent, et la façon dont ces messages peuvent être utilisés. Nous avons souligné que la variété de BASIC employée dans cette classe produit des messages d'erreur en Français à la compilation et à l'exécution, ce qui peut constituer un progrès sur des langages où les messages sont simplement numérotés, ou exprimés en Anglais; cependant, la lecture des comptes rendus montre que le rapport entre les messages d'erreur et le défaut de conception qui a conduit l'élève à l'écriture erronée, est souvent lointain; ou, de façon plus précise, la mise en évidence du rapport entre le message d'erreur, et l'erreur de conception suppose une connaissance du domaine que les élèves n'ont évidemment pas, puisqu'ils sont en apprentissage.

Les particularités de BASIC par exemple l'initialisation automatique d'une variable dès que son identificateur est rencontré, et la représentation des booléens par des variables numériques contribuent à cette distance entre le message et le défaut de conception: ainsi BASIC ne peut produire le message "*identificateur inconnu*", qui serait pourtant fort utile, ne serait-ce que pour déceler des erreurs d'orthographe dans un identificateur; de la même manière, nous avons vu par exemple qu'une expression conditionnelle erronée telle que `A=B=C` peut être interprétée par BASIC comme l'égalité de l'expression booléenne `A=B` et de la variable numérique `C`, et donc ne pas entraîner d'erreur à la compilation. Dans d'autres langages, des difficultés analogues auraient été rencontrées; ainsi, `A`,`B` et `C` étant des identificateur de variable d'un autre type que les booléens, la condition `A=B=C` produit une erreur de type sans rapport avec le défaut de conception qui conduit à cette écriture.

Par ailleurs, un message d'erreur à la compilation ou à l'exécution résulte d'une analyse trop locale du programme, pour permettre un diagnostic sur l'erreur de conception qui l'a produite; ainsi, les erreurs d'initialisation, dans les problèmes **PIANO** et **FILE** sont décelées seulement lors du premier appel de la fonction sous-chaîne, et cet appel peut être situé loin de l'initialisation dans le programme. Ceci conduit le plus souvent l'élève à considérer le message d'erreur comme l'indication que "*quelque chose ne va pas*", sans utiliser le message pour préciser ce "*quelque chose*"; la réponse du dispositif est ainsi sérieusement appauvrie. Ceci peut conduire à des améliorations syntaxiques visant à la disparition de l'erreur, sans que cette erreur ait été analysée: nous avons relevé par exemple que les contraintes syntaxiques liées au caractère fonctionnel de la fonction sous-chaîne (`MID$` en BASIC) peuvent être intégrées par certains élèves, sans que le résultat de la fonction ait un sens pour eux. Les réponses du dispositif en terme de messages d'erreur peuvent donc conduire à ce qu'une contrainte soit prise en compte par un élève dans ses écritures, sans que la logique qui la fonde soit comprise de cet élève.

---

[129] Dans cette situation, le "milieu a-didactique" se distingue bien de l'enseignant; il pourrait ne pas en être de même dans deux autres types de situations: les situations où le dispositif est "simulé par l'enseignant" (par exemple [LABORDE, BALACHEFF, MEIJAS 1985]), et celles où le dispositif sert à l'implémentation des solutions et simultanément sert de "tuteur intelligent" (en guidant l'élève vers la maîtrise des notions) ou de "détecteur d'anomalies" (en comparant la solution de l'élève à des déviations de schémas de solution) . Pour une revue des recherches concernant ce second type de situation, voir [LUCCI 89] ou [du BOULAY SOTHCOTT 87]

Le second type d'effets en retour du dispositif est constitué par les **résultats d'exécution**: les comptes rendus montrent que, le programme ne présentant plus d'erreurs syntaxiques, des résultats d'exécution erronés sont pris en compte de façon différenciée par les élèves pour améliorer ce programme; ainsi **Arnaud-Jérôme** ne parviennent pas à expliquer leur résultat erroné dans la résolution de **TESTC2** (séance 4); ils ne mettent pas en évidence que l'erreur résulte du calcul de la position du caractère milieu, et il faut plusieurs interventions de l'enseignant pour que l'erreur soit corrigée; **Natacha-Karine**, par contre, corrigent seules leur calcul de la position du caractère milieu, en fonction du résultat erroné qu'elles obtiennent à partir de leur expression spontanée de cette position; le début de compréhension qu'ont les élèves joue évidemment un rôle important dans le caractère producteur des résultats d'exécution; l'interprétation des résultats d'exécution non conforme à l'anticipation participe donc au processus d'acquisition.

L'amélioration du programme obtenue suite à la prise en compte des résultats d'exécution conduit-elle les élèves à progresser dans leurs acquisitions? Dans la résolution de **TESTC2** (séance 4), l'expression finale obtenue pour la position du caractère milieu est `L/2+0.5`; cette expression suffit pour que le programme soit juste[130]; elle prend en compte le fait que `L` est un nombre (impair); les élèves sont donc conduits à considérer le statut numérique de la longueur de la chaîne, ce qui constitue un progrès par rapport à leur conception de départ; cependant, l'ajout de `0.5` est une amélioration locale, qui ne donne pas aux élèves une meilleure compréhension de la notion de milieu de deux ordinaux; **Karine-Natacha** tentent d'ailleurs d'appliquer la même correction pour améliorer la condition d'arrêt dans **TESTC3**, alors que cette correction ne se justifie pas dans ce cas. Dans d'autres problèmes, le progrès peut être plus net; par exemple, concernant l'articulation entre variables et constantes dans une même écriture (**INSEE3**, séance 8). Par contre, concernant l'articulation de structures algorithmiques comportant plusieurs instructions, qu'il s'agisse d'alternatives ou d'itérations, les améliorations sont apportées le plus souvent suite à des interventions importantes de l'enseignant; elles semblent peu influencer les conceptions des élèves.

8.3 L'intervention de l'enseignant auprès des élèves

Le rôle de l'enseignant dans des séances de résolution de problème en informatique est certainement un des aspects essentiels de la situation; nous avons centré notre étude sur les élèves, leur rapport au problème et au dispositif, c'est pourquoi, nous disposons de peu d'hypothèses et d'observations pour analyser la nature et la pertinence des interventions du professeur et de nos interventions en tant qu'observateur. L'intervention enseignante dans la situation observée s'appuie sur une connaissance plus ou moins approfondie des S.R.T. que l'élève met en oeuvre, à partir d'indices qu'il peut tirer de l'observation de l'élève:

- l'élève n'exprime pas ses intentions et encore moins les représentations des objets et/ou des traitements qui guident ses intentions (son(ses) plan(s)). L'enseignant doit donc les inférer à partir de son observation des éléments de programmes écrits par l'élève dans le langage de programmation, en particulier des expressions qui lui semblent erronées.
- contrairement aux messages d'erreur du langage de programmation, l'analyse de l'enseignant met en rapport une expression isolée avec le reste du programme. Mais elle peut se révéler elle-même trop locale pour une intervention adaptée. En effet:
    - l'enseignant est, particulièrement au début de la résolution, dans l'ignorance des intentions des élèves, ce qui peut le conduire à des erreurs d'interprétation: une écriture qui lui paraît, sans signification de façon isolée, peut prendre son sens après que l'élève ait écrit la suite de son programme. Le *plan* de l'élève se révèle à lui seulement progressivement.
    - nous avons vu par exemple, dans l'étude des difficultés de **Karine**, qu'une expression comme `LET A$=MID$(LETTRE$,x,1)` dans **CRYPTAGE** (problème 3 de l'épreuve sur papier), prend son sens seulement après la

[130] En fait, le programme n'est juste que parce que `L` est de type réel, ce dont les élèves n'ont pas conscience, à cause de la déclaration implicite du type réel en BASIC.

mise en évidence de la représentation "incantatoire" de l'affectation par cette élève; ne disposant pas de cette connaissance des représentations de l'élève, nous n'avons pu la guider efficacement au cours de sa résolution des problèmes de "recherche-translation". De même, la mise en évidence pour deux élèves, d'une représentation des objets comme entités physique non duplicables suppose l'analyse d'une différence de comportement dans deux problèmes proches (**FILE** et **ONJOURB**). Ainsi, une analyse de la signification des écritures erronées d'un(e) élève peut nécessiter un nombre important d'observations de ces écritures, face à des problèmes variés.

Conclusions de la partie III

Les séries de problèmes présentées au chapitre 9 ont été conçues en fonction des difficultés dûes à l'existence de représentations des objets informatiques non encore suffisamment différenciées des représentations des objets dans le contexte familier, et de l'interaction de ces représentations avec les représentations des traitements. De façon à délimiter la tâche de l'élève, et l'effet attendu de cette tâche sur les représentations des élèves, nous avons proposé une classification en 4 types de problèmes:

- dans les problèmes (*CALCUL ARGUMENTS*) les élèves ont seulement à comprendre la structuration des données, et à résoudre les questions de mise en oeuvre des fonctions
- dans les problèmes (*CONSTRUCTION DE FONCTIONS*), une représentation des données à l'aide des chaînes de caractères étant proposée dans l'énoncé, les élèves doivent concevoir les fonctions sur ces données nécessaires à la résolution, et pour cela, composer des fonctions élémentaires en ayant recours à l'affectation, et/ou à la notation fonctionnelle,
- les problèmes (*CONSTRUCTION ITERATION*) sollicitent la coordination entre représentations des objets et représentations des traitements non séquentiels, tout en restant à un niveau de complexité compatible avec une activité de résolution autonome des élèves: il s'agit soit des problèmes classiques de comptage d'occurrence, soit des problèmes de "recherche-translation" où le programme doit rechercher la position d'une chaîne donné dans une chaîne constante, puis en opérant une translation dans cette chaîne, calculer la chaîne résultat.
- dans les problèmes (*CHOIX TYPE*), les élèves ont à faire le choix d'une représentation des données, en fonction des opérations que l'on souhaite effectuer sur ces données (et donc d'une abstraction implicite), plutôt qu'en fonction de leur apparence externe. La nécessité d'utiliser des fonctions de conversion apparaît également à cette occasion.

L'observation de la résolution guidée de ces problèmes, par 4 élèves particulièrement en difficulté, appuyée par les réponses de ces élèves à une épreuve sur papier, nous permet de tirer des conclusions sur l'évolution de ces difficultés, de comprendre les raisons de la persistance de certaines représentations, de critiquer a posteriori l'analyse ayant conduit à la conception des séries, et de situer certains éléments de la situation d'apprentissage que constitue cette résolution guidée.

1 L'évolution des élèves

1.1 Le typage des objets

Les élèves que nous avons observés, **surmontent la difficulté *D.TYPE*** (difficulté à envisager le **type** des objets en fonction des opérations nécessitées par le problème, et non en fonction de leur aspect externe). Ce progrès a une influence positive sur leur représentation des objets informatiques, et **facilite leurs acquisitions en informatique**: la série INSEE, centrée sur des problèmes (*CHOIX TYPE*) marque en effet un moment décisif dans la progression des élèves.

1.2 Calcul et affichage

Nous avions relevé au chapitre 6 que des élèves pouvaient, en pratiquant l'affichage successif de résultats partiels, ne jamais employer la concaténation des chaînes. Ce point nous avait paru très négatif, d'une part parce que la concaténation est une opération importante, représentative d'opérations sur des objets non-numérique, et que par conséquent, il est essentiel que les élèves s'en construisent des représentations adaptées, d'autre part parce qu'un programme informatique doit être envisagé comme une suite de calculs sur des données, et non comme une suite d'actions sur un terminal. Nous avons montré que l'affichage successif des résultats partiels contribue à une conception où l'instruction de sortie est comprise comme agissant sur

l'état interne du dispositif et qu'il peut mener à des impasses dans la programmation.

Lors de l'observation, nous avons imposé l'exigence du calcul du résultat et son affectation à une variable, comme préalable à son affichage. Cette exigence, motivée par le fait que le résultat d'un programme doit pouvoir être utilisé dans un autre programme, a eu un effet positif en conduisant les élèves à utiliser la concaténation, et l'affectation plutôt que l'affichage. Ce faisant, les représentations mentales de cette opération et de cette action sont intervenues, et ont évolué comme nous le voyons ci-dessous.

1.3 La représentation incantatoire de l'affectation

Suite à plusieurs de nos choix (affichage seulement en fin de calcul, comme rappelé ci-dessus, affectation des résultats partiels lors de la construction de fonctions composées), les élèves observés utilisent largement l'affectation pour les résultats intermédiaires, et les difficultés, sont nombreuses lors des premières séances, particulièrement dans les problèmes (*CONSTRUCTION DE FONCTIONS*). Elles sont liées à une représentation où les notions d'affectation et de condition se confondent dans une compréhension "incantatoire": l'affectation code un état que l'élève souhaite voir réalisé, et non un mécanisme par lequel la valeur d'une variable évolue. Par exemple, l'affectation syntaxiquement correcte: `LET L$=MID$(MOT$,X,1)` peut pour l'élève désigner `X` comme le rang du caractère `L$` dans la chaîne `MOT$`, au lieu d'être comprise comme une instruction donnant une valeur au caractère `L$`[131].

La représentation incantatoire de l'affectation est de façon évidente liée à la forme d'itération `REPETER...JUSQUA <condition>`. En effet, `<condition>` a souvent une forme syntaxique proche de l'affectation, et elle est valide à la sortie de l'itération. Comme l'élève n'a pas conscience du mécanisme itératif qui permet, par la (les) affectation(s) de variables itératives dans le corps de boucle, la validité de cette condition, *il pense que l'écriture de la condition (ou de l'affectation avec laquelle il la confond) suffit à provoquer sa validité.*

L'affectation, lorsque l'expression à affecter ne fait pas intervenir la variable sur laquelle porte l'affectation, est *compatible* avec cette représentation "incantatoire" de l'affectation, c'est-à-dire qu'une écriture conforme aux règles de l'affectation n'entre pas en contradiction avec cette représentation. De même, comme nous l'avons vu ci-dessus, la condition de sortie d'une itération, lorsqu'elle exprime qu'un résultat a été obtenu, s'apparente pour l'élève à une affectation comprise de façon "incantatoire". Il est donc probable que les premières utilisations de l'affectation et des conditions de sortie d'itération ne font que renforcer cette compréhension.

La représentation incantatoire de l'affectation persiste seulement chez une élève: il est remarquable que, chez cette élève en fin d'apprentissage, cette représentation de l'affectation se rencontre seulement en interaction étroite avec une représentation erronée de l'itération. Nous avons noté de façon générale une influence positive de la fréquentation d'un type non-numérique qui oblige les élèves à des différenciations par rapport à leur représentation intuitive des objets, et donc clarifient le mécanisme par lequel ils prennent des valeurs.

1.4 Les difficultés avec l'affectation: artefact du choix du langage, ou intrinsèques à l'activité de programmation?

On pourrait déduire de l'observation des difficultés que rencontrent les élèves à

---

[131] Cette représentation "incantatoire" ne doit pas être confondue avec une autre, rencontrée au chapitre 6 où l'affectation n'est pas comprise comme une action, l'identificateur de variable en première partie n'ayant pas de sens et n'étant pas réutilisé: dans cette dernière représentation, c'est *l'écriture fonctionnelle* qui est comprise comme modifiant l'état du dispositif, les éléments syntaxiques de l'affectation étant présents seulement pour la correction syntaxique: ainsi, dans cette représentation, l'écriture ci-dessus aurait pour fonction de "séparer le Xième caractère", sans que la variable `L$` soit comprise comme prenant ce caractère pour valeur (voir ci-dessous: *le caractère fonctionnel de de la fonction sous-chaîne*).

se construire une représentation adaptée de l'affectation, que ces difficultés sont introduites de façon artificielle par le choix d'un langage impératif, qui impose le recours à l'affectation, et que par conséquent, un style de programmation fonctionnel, ou une forme de programmation déclarative s'adapteraient mieux aux représentations spontanées des élèves.

Pour nous, avec les difficultés liées à l'affectation, ce sont les *contraintes du dispositif* que rencontrent les élèves, et la construction de systèmes de représentation adaptés suppose que ces contraintes aient été rencontrées et intégrées. Il nous semble donc que ces difficultés, liées aux caractéristiques générales des dispositifs informatiques plus qu'à un style de programmation, apparaîtront nécessairement sous d'autres formes dans des styles de programmation n'utilisant pas l'affectation. Par exemple, la programmation "déclarative" ne saurait se contenter d'une représentation "incantatoire": on ne peut programmer en déclaratif si l'on n'a pas conscience de l'algorithme d'effacement, à l'oeuvre dans l'évaluation des expressions.

1.5 La concaténation et la représentation des objets comme entités physiques non-duplicables

Le calcul d'une partie de la chaîne définie comme un "reste", dans deux problèmes de type (*CONSTRUCTION DE FONCTIONS*) fait apparaître chez certains élèves observés des comportement différents dans deux problèmes pourtant conçus comme similaires: dans un problème où il s'agit de mettre à jour une chaîne représentant une file, après dépilage d'un caractère, les élèves se posent explicitement et à juste titre la question de l'élimination du caractère, alors même qu'ils ont calculé ("extrait") ce caractère à l'aide de la fonction `MID$` en vue de l'afficher. Au contraire, dans le problème **ONJOURB** (où il s'agit de "passer à la fin" le premier caractère d'un mot), deux élèves se contentent de concaténer dans l'ordre ce mot et son premier caractère qu'ils ont antérieurement calculé et affecté à une variable. Nous pensons que les conduites de ces élèves face aux deux situations ne peuvent s'expliquer que par une représentation où les données introduites dans l'ordinateur seraient des objets non duplicables: l'affectation à une variable d'un élément calculé d'une chaîne, par exemple, ne conduirait pas à la duplication de la valeur de cet élément, mais à ce que la variable représente effectivement cet objet. Cette représentation n'empêche pas dans le premier problème un processus de résolution correct, mais conduit à une réponse erronée dans **ONJOURB**; en effet, **ONJOURB** fait intervenir la *concaténation de deux éléments issus d'une même chaîne.* Si une partie calculée de la chaîne ne peut être dupliquée, sa *concaténation* à cette même chaîne ne peut que la *déplacer.* Par contre, la concaténation d'éléments provenant de chaînes différentes, à partir du moment où elle est clairement distinguée de l'affichage, ne semble pas poser de problème pour les élèves observés: ici aussi, les régles régissant la concaténation sont *compatibles* avec la représentation erronée.

1.6 Le caractère fonctionnel de de la fonction sous-chaîne

Les contraintes syntaxiques liées au caractère fonctionnel des expressions utilisant la fonction `sous-chaîne` (`MID$` en BASIC), sont intégrées à l'issue de l'observation; cette intégration s'accompagne généralement d'une compréhension du caractère fonctionnel de `MID$`. Chez une élève, dans des circonstances particulières, le caractère fonctionnel peut être oublié, la syntaxe restant correcte. Seule une des élèves observés produit des écritures où le premier argument est erroné, la chaîne dont le résultat est "extrait" étant alors, pour elle, implicite; cette difficulté est liée aux difficultés de l'élève concernant *l'affectation*, et également à la *représentation des objets comme entités physiques non duplicables* (Cf ci-dessus).

1.7 La compréhension des arguments

Le couple (second argument , troisième argument) est parfois compris comme un ensemble ayant pour fonction de donner "toutes les informations nécessaires" à la détermination de la sous-chaîne, dans une représentation où le langage peut être utilisé de façon naïve: les informations données en argument ne sont pas dans ce cas compris comme permettant le *calcul* de la sous-chaîne, mais comme un ensemble vague

d'indications incluant des éléments d'ordre sémantique. Cette représentation apparaît de façon transitoire chez les élèves particulièrement dans les problèmes (*CALCUL ARGUMENTS*). Ici aussi, le couple des arguments dans la définition est *compatible* avec la représentation erronée: en effet il donne bien "toutes les informations" permettant de définir une sous-chaîne. C'est pourquoi cette représentation peut ne pas être remise en cause par certains exemples ou exercices: il y a bien nécessité, pour mettre au jour ce type de représentation, de problèmes sollicitant fortement le passage d'un niveau intuitif au niveau du calcul.

1.8 Les ordinaux

La translation (ajout d'un cardinal à un ordinal) est acquise par les élèves comme un moyen de résoudre des problèmes, après l'abandon de formulations ne prenant pas en compte les contraintes du dispositif. Au cours de la résolution des problèmes les élèves ont rencontré des difficultés liées au calcul des ordinaux et les ont résolues; à l'épreuve sur papier, la réponse d'une élève comporte cependant une écriture erronée, dont la confusion entre un ordinal et la position qu'il occupe dans une chaîne est partie prenante.

1.9 Les problèmes (*CONSTRUTION ITERATION*) et l'articulation objets-traitements

Le cas d'une structure alternative avec une condition complexe portant sur des chaînes est résolu assez facilement; l'articulation d'alternatives complexes est plus difficile pour les élèves, mais les difficultés ne paraissent pas spécifiques aux chaînes de caractères.

La résolution d'un problème du type "**comptage d'occurrences**" fait apparaître les difficultés classiques de l'itération. Parmi celles-ci, ce sont essentiellement l'évolution des paramètres de boucle, et la construction de la condition d'arrêt qui interagissent avec les difficultés liées à la structure de chaîne.

Dans les problèmes du type "**recherche-translation**" les élèves rencontrent la difficulté repérée et expliquée dans EPR2: la nécessité d'une itération pour obtenir la position de la lettre donnée, ne leur apparaît pas, et il y a confusion entre cette position et le caractère lui-même. Les élèves rencontrent d'autres difficultés qui tiennent à la séparation du programme en deux modules, le résultat du premier module étant la donnée du second. Cette séparation résulte du choix de la structure itérative `REPETER...JUSQU'A...` qui a dû être employée d'emblée par les élèves, contrairement à ce qui était envisagé lors de l'analyse préalable. Nous en examinerons ci-dessous les conséquences.

Nous avons pointé dans la partie I, **l'articulation objets-traitements** comme un élément épistémologique important dans l'activité de programmation, et comme un enjeu psychologique essentiel dans les premiers apprentissages. Parmi les problèmes (*CONSTRUTION ITERATION*) les problèmes de "comptage d'occurrences" et les problèmes de "recherche-translation" font intervenir de façon différente cette articulation: les premiers sollicitent la compréhension de la structuration des données sur des points précis de la construction de l'itération (paramètres de boucles, condition de sortie), alors que dans les seconds, c'est au niveau du plan d'ensemble du programme qu'il convient de prendre en compte les contraintes de la structuration des données. Les problèmes de "comptage d'occurrences" apparaissent donc surtout comme une complexification des problèmes (*CONSTRUCTION de FONCTIONS*), alors que les problèmes de "recherche-translation" impliquent une remise en cause en profondeur des conceptions du dispositif. Voyons si les élèves observés parviennent à cette remise en cause.

Certains progrès sont constatés à l'issue des séances de résolution, par exemple, le succès d'un des élèves observés à l'épreuve sur papier dans un problème de "recherche-translation", succès qui, lors de EPR2, n'avait été constaté que chez les élèves les plus brillants. Par contre, les éléments qui y conduisent n'ont pas de caractère d'évidence: nous n'observons pas de façon évidente les effets d'un déséquilibre dans les représentations des objets qui amènerait une réequilibration dans les représentations des traitements; une meilleure représentation des objets

s'observe, comme nous l'avons vu ci-dessus, mais ses effets sur les représentations des traitements apparaissent diffus. C'est un des points faibles de l'analyse préalable, qui nous conduisent ci-dessous à un retour sur cette analyse.

2 Des représentations avec lesquelles les régles du dispositif sont *compatibles*

D'une façon générale, cette nouvelle observation, moins ponctuelle que celle de la partie II, montre que le champ d'étude que constituent les représentations des objets du type "chaîne de caractères" reste largement ouvert, puisque de nouveaux aspects apparaissent ici: représentation "incantatoire" de l'affectation, représentation des objets comme entités physiques non duplicables, compréhension des arguments de la fonction sous-chaîne. Nous avons noté que ces représentations présentent une caractéristique qui les rend *résistantes*: les réponses correctes à certains exercices, peuvent s'obtenir malgré ces représentations; de même certains exemples donnés par le professeur peuvent ne pas mettre en cause ces représentations. En fait la représentation relative à un élément du langage (affectation, fonction...), à l'oeuvre chez l'élève, est, dans ce cas, une conception trop générale pour être opératoire dans toutes les situations, mais conduit, sous certaines conditions, aux mêmes écritures que les régles du langage. Nous disons que les régles du langage sont *compatibles* avec ces représentations: des réponses conformes aux régles n'entreront généralement pas en contradiction avec ces représentations; par contre, sous certaines conditions seulement, des réponse déduites par l'élève de ces représentations pourront entrer en contradiction avec les régles du langage. Il est clair que si les exemples et exercices proposés aux élèves ne prennent pas en compte en compte l'existence de ces représentations, ils peuvent en fait les renforcer.

3 Retour critique sur l'analyse préalable.

3.1 La place des problèmes de "recherche-translation" dans la progression.

Les problèmes de "recherche-translation" sont ceux qui permettent le mieux de faire intervenir les interactions entre les représentations des traitements et les représentations des données: cette hypothèse testée au chapitre 7, s'observe ici sur un plus grand nombre de problèmes, et par comparaison avec les problèmes de "comptage d'occurrences". Nous avions pensé, à la suite du chapitre 7, qu'une remise en cause des représentations des objets pouvait conduire à une représentation plus adaptée du traitement, ce qui ne s'observe pas ici, au moins au cours de la résolution des problèmes spécifiques. En fait, les difficultés des élèves à obtenir une solution correcte de ces problèmes nous ont surpris: nous pensions que, dans la première série, le premier problème de "recherche-translation" (ALPHA4) serait facilement résolu par "concaténation" des solutions de ALPHA2 (qui consiste à opérer une *translation*) et de ALPHA3 (qui consiste en la *recherche* du rang de la donnée). Nous n'avons pu repérer suffisamment tôt que la représentation "incantatoire" de l'affectation, *compatible* avec une solution de ALPHA3 sous la forme `REPETER...JUSQU'A,` constitue un obstacle de taille pour la résolution de ALPHA4. N'ayant pas caractérisé jusqu'ici ce type de représentation, nous n'avons pu agir dessus au cours de nos interventions auprès des élèves.

Ce défaut d'analyse, apparu à partir de la mise en oeuvre de la forme d'itération `REPETER...JUSQU'A.`, en fait apparaître un autre: l'analyse préalable postulait que les élèves peuvent exprimer leurs solutions d'abord à l'aide de la forme `POUR...FINPOUR`, et passer ensuite à la forme `REPETER...JUSQU'A...` Nous avions ainsi prévu que les élèves utiliseraient la forme `POUR...FINPOUR` dans les problèmes de la série ALPHA, puis passeraient à la forme `REPETER...JUSQU'A.`pour résoudre PIANO ou TESTCHA; il aurait alors été possible de montrer que la forme `POUR...FINPOUR` est la plus efficace, en revenant sur la série ALPHA. Nous avons expliqué que cette démarche est apparue irréaliste dès les premières séances: pour que

des élèves puissent passer d'une forme d'itération à une autre, il est nécessaire que l'itération soit installée chez eux à un niveau assez général (c'est-à-dire que les représentations liées à ces diverses formes soient mises en relation). Les difficultés des élèves observés résultant du caractère peu différencié et peu hiérarchisé de leurs représentations, il n'est pas étonnant qu'ils ne peuvent envisager deux formes d'itération simultanément: seule la forme la plus récente `REPETER...JUSQU'A...` a, de ce fait, pu être utilisée avec les élèves.

D'une façon générale, il aurait fallu d'une part considérer que des débutants présentant des difficultés d'acquisition ne peuvent envisager simultanément deux formes d'itération. D'autre part, pour que la résolution des problèmes de "recherche-translation" conduise aux effets souhaités, il aurait fallu que les représentations des objets (et en particulier des ordinaux et de l'affectation) aient connu chez les élèves, un début de différenciation, et aussi que nous les ayons caractérisées de façon à pouvoir agir sur elles dans nos interventions. Ceci implique de ne pas grouper ces problèmes en début de progression. En particulier, il aurait été intéressant d'observer chez **Karine** la résolution d'un problème de "recherche-translation" après la série **INSEE**, à un moment où, nous semble t'il, ses représentations ont sensiblement évolué.

3.2 La représentation des objets informatiques comme entités physiques non duplicables.

Seulement entrevue dans les observations qui ont précédé la rédaction des séries de problèmes, cette difficulté n'a pas fait l'objet d'une analyse préalable et d'une construction de problèmes en conséquence; il serait certainement souhaitable d'inclure à cette fin un (des) problème(s) où les élèves auraient à changer l'ordre des caractères d'une chaîne.

3.3 Les opérateurs de comparaison:

Leur emploi sur des chaînes de caractère représentant des nombres dans des problèmes (*CHOIX TYPE*), ne met pas en évidence qu'il s'agit d'une comparaison selon l'ordre alphabétique étendu au jeu de caractères de l'ordinateur, et non selon l'ordre dans les nombres entiers; cette mise en évidence aurait pourtant été féconde, du point de vue de la représentation des objets par les élèves.

3.4 Point forts et points faibles

Le travail d'ingénierie didactique s'est appuyé utilement sur l'élucidation des représentations (en particulier la conception naïve du langage, les conceptions des ordinaux, l'absence de prise en compte du typage...) apparues au chapitres 6 et sur l'analyse de la résolution de problèmes de recherche-translation au chapitre 7: l'intérêt d'un tel travail apparaît lorsque l'on constate que ces trois élèves, très en difficulté au départ, ont des résultats à l'épreuve sur papier honorables face aux élèves ayant passé EPR1 et EPR2. Les faiblesses de ce travail résultent de l'existence, chez les élèves, de représentations non encore mises en évidence, mais aussi d'une insuffisance d'analyse portant sur les points suivants: place des problèmes de "recherche-translation" dans une progression, niveau des représentations nécessaires pour comparer deux formes d'itération, implications cognitives de la comparaison des chaînes.

4 La situation pédagogique: moyens de l'élève, interventions de l'enseignant.

La situation conduit dans une certaine mesure les élèves à tenter des **formulations en langage naturel**, nécessitées par la résolution du problème, et non par des exigences didactiques; elle ne fait cependant pas dépendre le succès de la production de formulations opérantes, ce qui serait certainement prématuré. L'étude de la communication dans ce type de situation pédagogique, du niveau de langage qu'elle utilise, de ses effets sur les représentations, nous semble ouvrir des perspectives de recherche, mais qui sortent du cadre de cette recherche.

L'environnement du problème participe au "milieu a-didactique" qui place l'élève en situation de déséquilibre et lui permet de progresser en trouvant des réponses à ce déséquilibre. Cependant, l'absence de production par les élèves de valeurs de test suffisamment généraux au cours de l'observation, montre que la **phase de dévolution** (qui, dans la situation que nous examinons, est réduite à la lecture de l'énoncé, et à une vérification de la compréhension de cet énoncé) demande à être développée. Il s'agit donc également une direction de recherche importante, l'objectif étant que les élèves s'approprient le problème dans sa généralité, en particulier par la capacité à produire un jeu de valeurs de test.

Les **effets en retour** que les élèves obtiennent de l'ordinateur participent également au "milieu a-didactique"; ce sont soit des messages d'erreur, soit des résultats d'exécution. Mais l'observation montre que les messages d'erreur à la compilation ou à l'exécution résultent d'une analyse trop locale du programme, pour donner à l'élève un diagnostic sur sa conception; la réponse du dispositif est ainsi sérieusement appauvrie. On peut donc chercher à améliorer la réponse du dispositif comme le font les systèmes "tutoriels" (voir note ci-dessus), en tentant de donner une interprétation plus fine de l'erreur à partir d'hypothèses sur la représentation ayant conduit à l'erreur,(par exemple, [LELOUCHE MALOUIN 91] proposent un "feed-back" à deux niveaux). Cependant, même améliorée, cette réponse du dispositif demande à être analysée par l'élève, or cette analyse suppose que l'élève puisse se représenter la façon dont le dispositif interprète l'écriture erronée, ce qui fait précisément partie des objectifs de l'apprentissage. C'est pourquoi, même avec une réponse du dispositif améliorée, nous pensons qu'une démarche pédagogique tendant à obtenir des élèves qu'ils interprètent les messages d'erreurs, est indispensable.

De même, **l'interprétation par l'élève de la réponse du dispositif, quand elle n'est conforme à son anticipation, ne va pas de soi**: elle constitue un moment important de l'acquisition. Elle semble particulièrement productive pour l'évolution des représentations que nous étudions, mais il existe peut-être des domaines comme l'articulation de structures algorithmiques complexes où elle n'est possible ni productive, et dans ce cas, il n'existerait pas d'autres moyens que de "programmer juste du premier coup".

Les formulations en langage naturel, l'interprétation des réponses du milieu sont des moyens pour l'élève de résoudre, et de progresser; nous avons vu qu'il utilise d'autant mieux ces moyens qu'il progresse dans ses acquisitions; mais parallèlement, ses progrès le conduisent à obtenir le programme, directement sans erreur. Ainsi, quand il aurait besoin de ces moyens, il ne peut en disposer; quand il peut en disposer, il n'en a plus l'utilité. La situation pédagogique que nous avons observée au cours des 9 séances conduit donc seulement de façon **transitoire** les élèves à utiliser ces moyens: c'est face à une réponse inattendue du dispositif que les élèves tentent d'expliciter le dysfonctionnement et d'interpréter la réponse.

L'expression en langage naturel du procédé de résolution, l'anticipation explicite des réponses du dispositif constitueraient des formes plus achevées d'emploi de ces moyens; elles garantiraient des acquisitions de plus haut niveau que l'expression d'une solution dans un langage de programmation que nous observons ici. Les situations pédagogiques qui permettraient aux élèves de progresser dans ce sens restent à définir.

L'intervention pédagogique de l'enseignant doit pouvoir se fonder sur une analyse des expressions produites par les élèves sur une période suffisamment longue, et sur des problèmes variés: l'"histoire cognitive" de chaque élève semble importante à prendre à compte, les analyses cliniques que nous avons menées montrent pour de nombreux élèves, la présence d'un élément central dans les diverses représentations qu'il(elle) met en oeuvre, dont la signification apparaît seulement par la mise en relation de nombreuses conduites dans des situations de résolution. Il nous semble par conséquent, que l'enseignant comme les concepteurs des "systèmes tutoriels" devraient prendre en compte cette dimension s'ils veulent atteindre une pleine efficacité.

L'intervention pédagogique, dans le cadre que nous avons observé, consiste en un

échange avec les élèves, à propos des expressions qu'ils produisent dans le langage de programmation au cours de leur activité de résolution des problèmes. Elle semble pertinente pour les questions que nous nous posons: la constitution de représentations adaptées des objets, et leur articulation avec les représentations des traitements. Les élèves rencontrent d'autres difficultés, en particulier l'articulation de structures algorithmiques, qu'il s'agisse d'alternatives ou d'itérations. Il nous semble que cette articulation suppose une conception suffisamment adaptée des objets et de leurs relations avec les traitements. Mais l'existence d'une telle conception n'est probablement pas suffisante, et il serait intéressant de faire d'autres hypothèses et d'autres observations.

Partie IV: Extension de la démarche de recherche au type booléen

Chapitre 11: Les objets de type booléen
Chapitre 12: Etude expérimentale en classe de première

Chapitre 11: Les objets de type booléen

Concept de booléen
Place des booléens dans les apprentissages en informatique
Les booléens dans les langages de programmation

Le type "booléen" constitue le second type de données dont nous étudions l'utilisation par des élèves de l'option informatique des lycées. Cette thèse a pour objet l'étude des représentations mentales que se font les débutants au cours de la résolution de problèmes de programmation, et du lien entre ces représentations et les acquisitions des élèves en informatique; il est donc nécessaire de se limiter à des données pouvant être représentées à l'aide de types simples (non structurés), c'est-à-dire essentiellement les chaînes et les booléens.

Nous avons, dans les chapitres antérieurs, montré que les élèves rencontrent des difficultés spécifiques, lorsqu'il abordent la programmation avec un type de données comme les chaînes de caractères, et que les difficultés qu'ils rencontrent dans l'utilisation des structures algorithmiques présentent des aspects particuliers dans ce contexte. Nous avons voulu examiner dans quelle mesure ceci se vérifie pour d'autres types de données ; c'est pourquoi nous abordons ici le type booléen. Il s'agit ici seulement de vérifier que la démarche conduite pour les chaînes de caractères peut se généraliser et non de mener une étude aussi approfondie que pour les chaînes de caractères ; nous examinerons des difficultés rencontrées par les élèves lorsqu'il abordent la résolution de problèmes impliquant le type booléen, et nous les mettrons en rapport avec des systèmes de représentation, sans chercher à être exhaustif, ni à construire une progression visant à remédier à ces difficultés.

Dans ce chapitre, nous tentons d'abord de cerner la signification de l'intervention du type booléen en informatique, comme nous l'avons fait au chapitre **5** pour les chaînes de caractères: nous examinons donc d'abord le contexte d'apparition d'une théorie des "booléens" (antérieure à l'informatique), puis nous discutons l'utilisation des booléens en informatique, en particulier nous distinguons deux axes possibles d'utilisation des booléens pour la codification d'informations. Nous faisons ensuite des hypothèses sur l'intérêt et les difficultés de l'introduction du type booléen auprès de débutants.

Au chapitre suivant (chapitre **12**), nous mènerons un examen de type clinique des difficultés spécifiques à ce type de données, et des représentations associées. Dans la conclusion de cette partie, nous résumerons les résultats de cette examen, et nous ferons une comparaison avec les résultats sur les chaînes de caractère.

1. Les booléens : contexte d'apparition d'une théorie des "booléens" ; utilisation en informatique

Nous examinons dans ce paragraphe :

- le contexte d'apparition de cette théorie, et ses applications actuelles.
- les définitions et théorèmes qui fondent "l'Algèbre de BOOLE",
- la place des booléens dans l'informatique, et son enseignement.

1.1 Le concept de booléen ; contextes d'apparition, et applications

«*Le but essentiel du mathématicien anglais Georges* BOOLE *(1815-1864) en fondant son Algèbre était de soumettre le raisonnement logique à des régles convenables de calcul*» ([CASANOVA 67][132]). Comme l'indique [HOFSTADTER 79][133]

---

[132][CASANOVA 67] Gaston Casanova L'Algèbre de Boole Que Sais-je? PUF 1967

[133][HOFSTADTER 79] D. Hofstadter (Bach Echer Gödel) Basic Books 1979 paru en traduction française InterEditions 1985

les efforts de BOOLE (et de De MORGAN) ont été motivés par la volonté de clarifier la notion de démonstration, en lien avec les recherches qui conduisent à la mise en place de la théorie des ensembles (HILBERT, CANTOR..). De tels efforts sont justifiés par la nécessité d'intégrer la découverte de géométries non-euclidiennes (LOBATCHEVSKI, RIEMANN...) elles-mêmes conçues au départ pour examiner les conséquences de la négation de l'énoncé d'EUCLIDE.

Aujourd'hui l'Algèbre de BOOLE existe comme chapitre de la logique, sous le nom de logique du premier ordre.(La logique du second ordre est caractérisée par l'emploi de variables et de quantificateurs). Mais elle existe également dans des domaines techniques pour des applications à première vue très éloignées de la logique: dans le même ouvrage G. CASANOVA détaille "l'Algèbre des circuits" comme une application de l'Algèbre de BOOLE à la résolution de problèmes concernant les circuits électriques, en énonçant le problème général :
«*Etant donné un dipôle de bornes* x *et* y *entre lesquelles se trouvent un nombre fini quelconque de contacts* A, B, C ...L *représentés par les variables de Boole* a,b,..,l, *trouver une fonction logique de ces variables, dite fonction de structure, qui soit égale à* 1 *lorsque le courant passera de la borne* x *à la borne* y, *et à* 0 *lorsqu'il sera interrompu.*»
Réciproquement, étant donnée une fonction logique, il est intéressant de développer des méthodes pour construire un circuit dont la fonction de structure soit égale à la fonction logique donnée. De telles méthodes trouvent un domaine d'application important dans différents domaines industriels pour la constitution d'automatismes à relais, pneumatiques, ou électroniques.

G. CASANOVA montre également comment l'Algèbre de BOOLE permet de résoudre par le calcul (sans raisonnement) des problèmes concernant les graphes : le problème de la connexité forte d'un graphe, c'est à dire l'existence, pour deux points quelconques, d'une suite d'arcs les reliant est résolu, en utilisant une matrice de booléens, et la multiplication matricielle pour les lois de conjonction et de disjonction : Soit `S` l'ensemble des sommets du graphe, et `PHI` la matrice de `SxS` vers `{V,F}` telle que `PHI(s1,s2) = V` si et seulement si `s1` est relié à `s2` dans le graphe, ou `s1=s2`. Soit `N` le nombre d'éléments de `S` ; le graphe a la propriété de connexité forte si et seulement si PHI^N est la matrice dont tous les éléments sont égaux à `V`.(voir en 1.3.3 la définition de la multiplication des matrices booléennes).

1.2 Les définitions et propriétés de l'Algèbre de BOOLE

Propositions :
soit `E` un ensemble; on appelle proposition toute application de `E` vers un ensemble à 2 éléments `{V , F}`.

Connecteurs logiques :

Conjonction : `p` et `q` étant deux propositions, on appelle conjonction de `p` et `q`, et on note $p \wedge q$ la proposition qui vaut V sur l'intersection des deux ensembles où respectivement `p` et `q` valent V.[134]

Disjonction : `p` et `q` étant deux propositions, on appelle disjonction de `p` et `q`, et on note $p \vee q$ la proposition qui vaut V sur la réunion des deux ensembles où respectivement `p` et `q` valent V.

Négation : `p` étant une proposition, on appelle négation de `p`, et on note $\neg p$ la proposition qui vaut `F` quand `p` vaut V, et qui vaut V quand `p` vaut `F`.

Fonctions logiques :
Une fonction logique associe à n propositions `p, q ,r` ... une proposition constituée à l'aide d'une combinaison finie des propositions `p, q , r`...reliés par des conjonctions,

---

[134] Dans les paragraphes suivants, lorsqu'il s'agira d'analyse informatique, nous utiliserons respectivement les formulations `P ET Q, P OU Q , NON P`)) pour désigner respectivement la conjonction, la disjonction des propositions `P` et `Q` et la négation de la proposition `P`. Dans un contexte d'emploi d'un langage de programmation nous utiliserons respectivement `P AND Q, P OR Q, NOT P`

des disjonctions ou des négations. Exemples :
q ∨ ¬p fonction à deux arguments, dénommée p implique q
r ∨ ¬q ∨ ¬p fonction à 3 arguments ; l'associativité est utilisée pour se dispenser des parenthèses.

Tautologies :
Elles expriment l'égalité de deux fonctions logiques :
les exemples les plus connus :

¬ ¬p = p
¬(p ∨ q) = ¬p ∧ ¬q

Elles permettent de transformer des fonctions logiques exemple : (r ∨ ¬q) ∨ ¬p se transforme en r ∨ ¬(p ∧ q) ce qui exprime l'équivalence logique entre (p implique (q implique r)) et ((p ∧ q) implique r).

Forme canonique disjonctive des fonctions logiques[135]
p, q, r ...étant n propositions il existe 2^n propositions formées de la conjonction de n termes distincts le premier égal à p ou à ¬p, le second à q ou à ¬q, etc...
exemple : pour trois propositions :

p ∧ q ∧ r p ∧ q ∧ ¬r p ∧ ¬q ∧ r
p ∧ ¬q ∧ ¬r ¬p ∧ q ∧ r ¬p ∧ q ∧ ¬r
¬p ∧ ¬q ∧ r ¬p ∧ ¬q ∧ ¬r

Ces propositions sont appelées "termes minimaux" ; ils sont exclusifs l'un de l'autre (c'est-à-dire que leur conjonction deux à deux a toujours la valeur F), et la disjonction des 2^n termes minimaux est la fonction dont la valeur est toujours V.
Propriété: toute fonction logique sur n propositions p, q, r,... peut se mettre de façon unique sous la forme d'une disjonction de termes minimaux formés à l'aide des p, q, r.
Exemple : forme disjonctive canonique de la fonction r ∨ (¬q∧¬p) :
(p ∧ q ∧ r) ∨(¬p ∧ q ∧ r) ∨(p ∧ ¬q ∧ ¬r)
∨(¬p ∧ ¬q ∧ r) ∨ (¬p ∧ ¬q ∧ ¬r)
Note: la forme initiale: r ∨ (¬q∧¬p) n'est pas la forme disjonctive canonique, en ce sens que les termes de la disjonction ne sont pas des termes minimaux; cependant, une telle forme, constituée d'une disjonction de termes formés eux-mêmes seulement de conjonctions et de négations, peut exprimer de façon utilisable une fonction logique; nous appelerons **disjonctive** une telle forme (c'est une forme disjonctive parmi d'autres; seule la forme **disjonctive canonique** est unique).
Applications : la forme disjonctive canonique est lourde, mais, pour une fonction donnée par un énoncé dans un problème, elle est facile à trouver : pour chacun des termes minimaux, on fait l'hypothèse que ce terme a la valeur V, et on examine la valeur de la fonction ; la forme disjonctive canonique s'obtient en faisant la disjonction des termes minimaux pour lesquels on obtient la valeur V.

Ainsi, ayant à construire, par exemple la fonction logique rendant compte de l'état d'un circuit électrique comme ci-dessus, et comportant 3 interrupteurs, on placera l'ensemble des interrupteurs successivement dans les 8 états possibles, et pour chacun de ces états, on examinera si le courant passe; la forme disjonctive canonique comprendra les termes minimaux correspondants aux états de l'ensemble des interrupteurs pour lesquels le courant passe ; on simplifiera ensuite éventuellement cette forme en utilisant les tautologies.

De façon pratique, **la table de vérité** d'une fonction logique exprime, pour chacun des termes minimaux, la valeur de la fonction quand on fait l'hypothèse que ce terme a la valeur V; elle est donc équivalente à la forme disjonctive canonique.

Exemple pour la fonction logique ci-dessus:

p	q	r	r∨(¬p ∧ ¬q)
V	V	V	V
V	V	F	F

[135] Une forme est dite canonique si, pour toute expression, il existe une expression équivalente, de cette forme, et une seule : par exemple, pour les polynômes, la forme réduite et ordonnée selon les puissances d'exposants croissants.

V	F	V	V
V	F	F	F
F	V	V	V
F	V	F	F
F	F	V	V
F	F	F	V

1.3 La place des booléens dans l'informatique et dans l'apprentissage de la programmation

1.3.1 La place des booléens dans l'architecture des ordinateurs et les langages de programmation

L'étude de textes fondateurs de l'informatique comme [VON NEUMAN 47][136] montre qu'avec l'EDVAC, la codification de l'information par un ensemble d'éléments physiques à deux états s'impose, en parallèle avec l'apparition d'une unité de commande basée sur des instructions logiques. La rupture s'effectue avec les calculatrices précédentes où dominent les chiffres décimaux (systèmes physiques à 10 états) et les instructions arithmétiques. De fait, les architectures des ordinateurs découlent des principes énoncés par VON NEUMANN, et en code machine, les instructions sont basées sur des opérations logiques opérant sur des "bits" regroupés en "mots" : `AND` (conjonction bit à bit de deux mots) `OR` (disjonction bit à bit de deux mots) `COM` (négation bit à bit d'un mots) , décalages ; les "instructions de saut" conditionnelles , qui permettent à ce niveau de constituer des structures conditionnelles sont elles-mêmes déterminées par la valeur d'indicateurs logiques ou drapeaux (indicateur de ZERO, positionné à 1 si la dernière instruction a produit un résultat nul, indicateur de SIGNE, positionné à 1 si la dernière instruction a produit un résultat négatif...).

Dans les langages évolués, les types numériques , et le type caractère dominent comme types de base, masquant au premier abord le rôle joué par les valeurs logiques ; en tant que type de données (valeurs et fonctions sur ces valeurs) les booléens sont nécessaires, au moins pour l'expression des conditions dans les instructions conditionnelles ; leur réalisation diffère selon les langages ; nous examinerons plus loin les langages les plus accessibles pour l'alphabétisation informatique.

1.3.2 L'emploi des booléens pour l'écriture des conditions

L'écriture d'expressions conditionnelles (alternatives, itérations, récursions) impose au programmeur de manipuler des quantités booléennes, consciemment ou non. En effet, tout langage, qu'il soit impératif ou fonctionnel permet la construction de structures conditionnelles (alternative, itération, récursion). La partie "condition" de ces structures est nécessairement de type logique. Le type de la condition peut être caché par une compréhension "naïve" de l'expression conditionnelle : a étant une variable numérique, la formulation `SI a=0 ALORS...` peut être comprise en première approche comme la traduction d'une phrase en langage naturel, et non comme une expression ayant un argument booléen dont la valeur est ici obtenue par l'évaluation de l'expression `a=0`. Mais des difficultés se présentent lorsque la condition devient complexe : donnons un exemple de ces difficultés : la formulation `SI a=0 OU a=1 ALORS...`pourrait également être comprise de façon naïve, comme une transposition d'une formulation en langage naturel ; le programmeur "naïf" ne comprend pas la nécessité d'une expression aussi lourde alors que la formulation spontanée en langage naturel est plutôt `SI a=0 OU 1 ALORS....`

A partir du moment où le programmeur est à même d'envisager consciemment la partie condition des expressions conditionnelles comme une quantité booléenne, il dispose de facilités pour l'écriture de cette partie : il pourra par exemple décomposer une condition complexe à l'aide d'affectations à des variables ayant des identificateurs signifiants ; dans l'exemple ci-dessus :

```
condition1 ← (a=0)
condition2 ← (a=1)
```

---

[136] [VON NEUMAN 47] J. Von Neumann First Draft on a report on the EDVAC 1945

```
condition ← condition1 OU condition2
SI condition ALORS ...
```

1.3.3 L'emploi des booléens en programmation pour la résolution de problèmes. Définition de deux axes

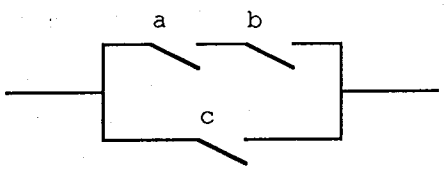
Nous distinguons deux axes généraux d'utilisation des booléens pour la résolution de problèmes ; pour chacun de ces deux axes nous pouvons prévoir un type de difficulté de résolution.

Axe 1 : les booléens pour la codification d'objets à deux états difficultés dans la construction des fonctions logiques sur ces objets

Nous avons énoncé au chapitre 4 (§1.2) des hypothèses sur les énoncés où les représentations spontanées du dispositif peuvent être conçues par l'élève comme adéquates, mais se révèlent d'une manière ou d'une autre, entrer en contradiction avec la logique de ce dispositif informatique. Nous avons dit en particulier que ces énoncés doivent imposer une "codification" d'objets intuitifs pour les élèves, sous forme de données informatiques et la production d'une solution comme "calcul" sur ces données codifiées

Si l'on s'intéresse au type booléen, on considérera donc des problèmes où les objets du monde réel peuvent prendre deux états : un interrupteur ouvert ou fermé, une barrière levée ou baissée... Dans ce cas les booléens s'imposent, en tant que type à deux valeurs ; de plus, les fonctions logiques de base (disjonction, conjonction, négation) permettent de représenter des situations élémentaires clairement identifiables : ainsi, pour représenter un circuit électrique comportant des interrupteurs, la fonction de conjonction, permet de représenter l'état d'un circuit composé de deux interrupteurs en série; la disjonction correspond à un circuit composé de deux interrupteurs en parallèle. Le lien entre les situations élémentaires et les fonctions logiques de base étant fait, le travail du programmeur consistera ensuite à préciser la(les) fonction(s) logiques nécessitées par le problème, et à les exprimer par un calcul utilisant les fonctions logiques de base. L'emploi des booléens permettra d'éviter tout recours à des conditionnelles, et offre donc davantage de lisibilité.

Dans ces problèmes, le programmeur qui ignore l'existence du type "booléens" pourra certes représenter les données par d'autres types, par exemple un type numérique ou le type caractère, en limitant à deux le nombre de valeurs prises par les variables, mais il devra recréer les fonctions logiques de base nécessaire dans le problème, en quelque sorte, recréer le type booléen, faute de quoi, il devra utiliser extensivement l'alternative. Considérons, pour illustrer cela, le problème consistant à exprimer l'état du circuit ci-dessous, connaissant l'état des interrupteurs



Un exemple de problème de l'axe 1: *Ecrire un programme prenant en entrée l'état des interrupteurs, et donnant comme résultat l'état ("passant" ou "non passant") du circuit.*

Trois formes de programmation:

1. La forme "alternatives complètes" : si l'on n'utilise pas les booléens, mais des chaînes de caractères et des alternatives, le programme peut prendre la forme suivante

```
SI c='fermé'
  ALORS circuit ←'passant'
  SINON SI a= 'ouvert'
            ALORS  circuit ←'non passant'
            SINON  SI b='ouvert'
                       ALORS circuit ←'non passant'
                       SINON circuit ←'passant'
                   IS
       IS
IS
```

2. La forme "remplacement conditionnel" : dans un style plus concis, on pourra éviter le recours à la partie SINON de l'alternative:

```
circuit ←'non passant'
SI c='fermé' ALORS circuit ←'passant'
SI a= 'fermé' ALORS  SI b='fermé' ALORS circuit ←'passant'
```

On remarquera que dans cette forme:

- l'alternative en chaîne permet de réaliser le ET logique.
- les alternatives successives permettent de réaliser le OU logique.
- le programme est plus lisible, mais demande davantage de maîtrise de l'affectation.

3. La forme "booléene" : Avec les booléens, la valeur V étant attribuée à l'état fermé pour les interrupteurs, et à l'état passant pour le circuit, le programme s'écrit :

```
circuit  ← c OU (a ET b).
```

4. Les formes mixtes: bien sûr tout "mélange de forme" est concevable; en particulier si les élèves ont rencontré le ET logique dans l'écriture des conditions, ils peuvent se passer de l'alternative en chaîne de la forme 2 et écrire:

```
circuit ←'non passant'
SI c='fermé' ALORS circuit ←'passant' IS
SI a= 'fermé' ET b='fermé' ALORS circuit ←'passant' IS
```

ou même:

```
circuit ←'non passant'
SI c='fermé'OU (a= 'fermé' ET b= 'fermé') ALORS
                                    circuit ←'passant' IS
```

Cette dernière forme pourra paraître peu différente de la forme "booléene". Elle suppose, comme la forme booléene, la maîtrise des connecteurs propositionnels, mais ces connecteurs portent ici sur des **conditions**, qui pour un élève *peuvent ne pas apparaître réellement comme des objets informatiques susceptibles par exemple de l'affectation.* Dans cette conception, l'écriture (a= 'fermé' ET b= 'fermé') est la traduction d'une forme verbale, et non un **calcul** sur des quantités informatiques.

Avantages et difficultés d'acquisition d'une forme de programmation "booléene".

Le gain en concision et en lisibilité est manifeste dans la troisième forme. Elle seule pourra permettre aux élèves d'aborder des problèmes plus complexes. Il est douteux que cette forme de programmation puisse être obtenue d'emblée des élèves en période d'alphabétisation informatique. Elle demande en effet de considérer les quantités logiques comme des valeurs au même titre que, par exemple les entiers ou les chaînes, de façon à faire porter sur ces quantités l'affectation et de considérer les connecteurs logiques comme des opérateurs sur ces valeurs, et non comme la traduction de formes verbales. Un précurseur facilitant serait peut-être un enseignement en logique reçu par exemple en mathématiques ou dans un domaine professionnel.

La logique propositionnelle a été supprimée des programmes de mathématiques de l'enseignement français depuis le début des années 1980, et il est donc douteux que les élèves de l'enseignement classique (non-professionnel) connaissent les notions de base de ce domaine. On peut penser que par contre, des élèves ayant reçu un enseignement portant par exemple sur des automatismes industriels (quelqu'en soit la technologie) ont acquis des notions leur permettant d'aborder plus facilement cette troisième forme ; en effet, dans ces domaines, les fonctions logiques de base sont identifiées à des dispositifs matériels ; un problème d'automatique consiste à réaliser un dispositif matérialisant une fonction logique donnée, donc à construire cette fonction, à l'aide des fonctions de base.

Axe 2 : problèmes où les booléens servent à représenter des relations logiques entre les données
difficulté à construire une codification adaptée pour ces relations logiques

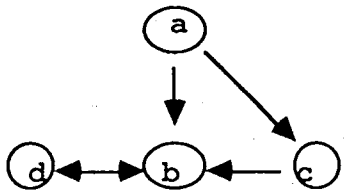
Les booléens permettent non seulement la codification des données du problème, mais aussi, en tant que type représentatif de données logiques de représenter des relations logiques concernant les objets ; par exemple la proposition "*Pierre aime la*

confiture" peut être Vraie ou Fausse ; elle exprime une relation entre des objets du programme.

C'est pourquoi, les booléens peuvent apparaître dans des problèmes de façon inattendue. Nous avons rapporté au chapitre 1 l'exemple fourni par [RSAC 83] dans le but d'illustrer la construction d'algorithmes par transformation de programmes : il s'agit de construire un algorithme de multiplication utilisant seulement l'addition et la multiplication par deux ; au moment de la dérécursivation, il est nécessaire de mémoriser la parité d'un opérande, ce qui conduit à l'utilisation d'une pile de booléens. L'utilisation des booléens est donc dans cet axe, un choix du programmeur dans le but de pouvoir considérer comme une donnée du programme certaines **relations logiques sur les données.**

Ce second type d'utilisation des booléens peut s'imposer plus difficilement au programmeur débutant ; il s'agit en effet de **passer d'une codification d'objets du monde réel à la codification des relations logiques sur ces objets** : les relations logiques doivent passer du statut d'outils (ce qui sert à résoudre le problème) à celui d'objet (ce sur quoi porte le problème).

- Un exemple de problème de l'axe 2 où l'utilisation de tableaux de booléens apporte davantage de généralité et de lisibilité. Un cas assez représentatif est celui des traitements sur les graphes. Considérons le problème suivant :



étant donné un graphe orienté, écrire un programme dont les données sont deux sommets, et le résultat l'existence ou non d'un parcours (suite d'arcs consécutifs) joignant ces deux sommets.

Si le graphe est simple, le programmeur peut éviter la question de la codification des relations logiques, et résoudre à l'aide d'alternatives. On suppose que `s1` et `s2` représentent respectivement le sommet de départ, et d'arrivée, et que `possible` est le résultat; cette variable prend la valeur `V` si un parcours existe. Le programme peut s'écrire dans une forme "mixte".:

```
SI s1=a ALORS possible ←V IS
SI s1=c ALORS SI s2=a ALORS possible ←F SINON possible ←V IS IS
SI s1=b ALORS SI s2=d ALORS possible ←V SINON possible ←F IS IS
SI s1=d ALORS SI s2=b ALORS possible ←V SINON possible ←F IS IS
```

Mais alors le programme est difficile à valider, et surtout ne présente aucun caractère de généralité. Il n'est pas possible de l'améliorer sous cette forme.

L'utilisation de tableaux de booléens permet l'écriture d'un programme indépendant du graphe considère : on peut en effet exprimer l'ensemble des sommets qu'il est possible d'atteindre par un trajet de longueur donnée, à partir de `s1`, comme un vecteur de booléens, `but_possible`, indexé sur les noeuds du graphe, et calculer ce vecteur itérativement, jusqu'à trouver `s2` parmi les sommets atteints, ou dépasser `N` (le nombre de sommets du graphe) comme longueur du trajet. Un trajet de longueur `N+1` repasse en effet nécessairement par un sommet, et est donc équivalent à un trajet de longueur inférieure.

Les éléments du vecteur `but_possible` ont pour valeur initiale `F`, sauf pour l'élément d'index `s1`.

On passe du vecteur représentatif de l'ensemble des sommets atteints par un trajet de longueur `I`, au vecteur représentatif de l'ensemble des sommets atteints par un trajet de longueur `I+1`, par la multiplication matricielle : `G * but_possible`, où `G` est la matrice du graphe, c'est-à-dire le tableau `[aij]` tel que `aij = V` si et seulement si le `i`ème sommet est relié au `j`ième sommet dans le graphe et `*` la multiplication matricielle pour les lois de conjonction et de disjonction.[137]

---

[137] La multiplication des matrices peut être définie pour des matrices dont les éléments appartiennent à un ensemble muni de deux opérations : si l'ensemble est un ensemble numérique, le produit d'une matrice-ligne (1,n matrice) par une matrice colonne (n,1 matrice) donne un élément, somme des produits des éléments correspondants de la ligne et de la colonne ; le produit d'une n,m

Les sommets étant dans l'ordre, a, b, c, d, la matrice du graphe ci-dessus s'exprime:

```
⌈V F F F⌉
|V V V V|
|V F V F|
⌊F V F V⌋
```

Si par exemple, s1 vaut c, le vecteur but_possible prend successivement les valeurs:

```
                ⌈F⌉                    ⌈F⌉                    ⌈F⌉
état initial    |F|   première itération |V|   seconde itération |V|
                |V|                    |V|                    |V|
                ⌊F⌋                    ⌊F⌋                    ⌊V⌋
```

ce qui indique que b est atteint à la première itération, et d à la seconde.

Le programme peut alors s'écrire :

```
I ←0
FAIRE
   I ←I+1;
   but_possible ← G * but_possible;
   SI but_possible[s2] ALORS possible ←V ; EXIT IS
   SI I= N ALORS possible ←F ; EXIT IS
BOUCLER
```

- Les difficultés d'emploi des booléens dans cet axe. Dans ce second axe d'utilisation des booléens pour la résolution de problèmes, les booléens interviennent pour représenter des relations logiques sur les objets du problème ; ces problèmes imposent de considérer ces relations elles-même comme des objets susceptibles de codification et de calcul. Un tel point de vue suppose a-priori une certaine maturité de la part des élèves, concernant le raisonnement logique, cette maturité devant permettre de passer du point de vue des relations logiques "*outils*" aux relations logiques "*objet*".
  On remarquera aussi que cette utilisation des booléens trouve sa pleine efficacité seulement avec l'utilisation de *types structurés* dont les éléments sont des booléens : pile de booléens dans l'exemple développé par [ARSAC 83], tableaux de booléens dans l'exemple ci-dessus.

2. Les booléens dans les langages de programmation

Nous avons souligné que tout langage de programmation doit disposer de booléens, ne serait-ce que pour l'expression de la partie "condition" d'une expression conditionnelle. Un langage doit disposer en premier lieu d'opérateurs de comparaison à valeur booléenne sur les types de base afin d'exprimer les conditions. Ces opérateurs sont l'égalité (sur tous les types), supérieur, inférieur, sur les types numériques, avant, après dans l'ordre alphabétique sur le type chaîne ou plus généralement sur les types énumérés. Il en est de même pour des outils de résolution de problème comme les tableurs (nous examinerons le cas de MULTIPLAN), et les gestionnaires de fichiers (nous examinerons le cas de DBASE).

L'existence du type booléen suppose de plus la possibilité d'affecter une valeur booléenne à une variable, ainsi que des opérations internes correspondant aux fonctions logiques élémentaires (négation, disjonction, conjonction)

Les principales différences d'un langage à l'autre portent sur l'existence des

---

matrice M par une m,p matrice P est la n,p matrice dont l'élément (i,j) est le produit de la ième ligne de M par la jième colonne de P.

Dans la multiplication de matrices booléennes qui nous intéresse ici, le produit est remplacé par la conjonction, et la somme par la disjonction ; ainsi, le produit d'une ligne et d'une colonne a la valeur Vraie si et seulement si il existe dans cette ligne et cette colonne des éléments de même rang ayant tous les deux la valeur Vraie.

booléens en tant que type spécifique, et sur la possibilité de faire opérer des entrées et sorties sur ces valeurs.

2.1 Le cas des langages où les booléens n'existent pas comme type spécifique

Dans les langages BASIC et LOGO[138], les booléens n'existent pas comme type spécifique ; les valeurs booléennes sont deux valeurs prises dans un type de base, et par conséquent, une quantité booléenne peut être impliquée dans une expression où elle est comprise non pas comme booléen, mais comme valeur du type de base.

2.1.1 Les valeurs booléennes en BASIC

En BASIC, un booléen est de type numérique (entier s'il est affecté explicitement à une variable de type entier, réel sinon).

Les opérations logiques (négation, disjonction, conjonction) sont définies pour les entiers : les entiers sont compris entre -32767 et + 32768 ; ils sont définis sur deux octets : un entier positif est représenté en mémoire par son écriture binaire , un entier négatif X, est représenté par l'écriture binaire de 65536 - X un opérateur logique correspond à l'opération logique élémentaire sur chacun des 16 bits de la codification binaire des opérandes :

3 AND 4 a pour valeur 0 (en binaire 0000 00011 AND 0000 0100)
NOT 3 a pour valeur -4 (en binaire 3 s'écrit 0000 00011
-4 s'écrit 1111 11100)

Ces opérations logiques sur les entiers servent dans les cas où ces entiers sont représentatifs de groupements de bits (octets, mots...). Par exemple, pour "forcer à 1" le bit 0 de la variable OCTET, on écrira:

```
LET MASQUE = 1
LET OCTET = OCTET OR MASQUE
```

Il est nécessaire que quelque soit la valeur de Y, X AND Y donne la valeur fausse si X est faux, , et X OR Y donne la valeur vraie, si X est vrai ; c'est pourquoi la valeur -1 (en binaire 1111 1111) est attribuée à toute expression booléennes vraie, et la valeur 0 (en binaire 0000 0000) à toute expression booléenne fausse.

A l'exécution, toute expression à valeur booléenne (le résultat d'une opération de comparaison, par exemple), est remplacé par la valeur 0 ou -1 ; il est donc possible de concevoir des expressions où sont impliqués des opérateurs à valeur logique, et des opérateurs à valeur numérique, par exemple:

```
(X=0)+2*(X=1)  a pour valeur   -1 pour X valant 0
                               -2 pour X valant 1
                                0 pour les autres valeurs
3 OR (X=0)     a pour valeur   -1 si X a pour valeur 0,
                                3 dans le cas contraire.
```

Cette possibilité est utilisée par certains programmeurs BASIC. Elle est par contre génératrice de confusions pour les apprentis-programmeurs non avertis de cette question de codification. Ainsi dans certains BASIC, l'expression IF X="A" THEN ACTION1 ELSE ACTION2, entraîne l'erreur "*opérateur de comparaison attendu dans une expression booléenne*" ; elle résulte en fait d'une erreur de type (X, variable numérique, "A" constante chaîne) dans la partie condition ; mais pour le compilateur, X variable numérique peut de façon valide être comparée à une expression à valeur booléenne ; le compilateur considère donc la constante chaîne "A" comme le début d'une expression à valeur booléenne, et attend l'opérateur de comparaison qui doit suivre pour former cette expression ; cet opérateur n'étant pas présent, il interrompt l'exécution, et affiche le message d'erreur rapporté ci-dessus.

2.1.2 Les valeurs booléennes en LOGO

En LOGO, les booléens, comme d'ailleurs les nombres, sont des noms, c'est à dire

---

[138] Ces langages, parmi d'autres possibles pour l'alphabétisation informatique sont présentés au chapitre 5, avec des références bibliographiques.

des objets dont l'aspect externe est une chaîne de caractères. En version française [LOGO 85], le nom attribué à la valeur vraie est "VRAI, le nom attribué à la valeur fausse est "FAUX ; contrairement à BASIC, les opérateurs logiques (NON OU ET) opèrent seulement sur les noms "VRAI et "FAUX : NON "ANATOLE produit l'erreur : "ANATOLE n'est ni vrai ni faux. De même la partie condition d'une expression conditionnelle doit avoir pour valeur le nom "VRAI ou le nom "FAUX.

Les fonctions et instructions sur les noms opèrent sur les valeurs booléennes ; ainsi :
PREMIER (0=0) a pour valeur le caractère V (PREMIER est la fonction qui à un mot fait correspondre son premier caractère).

Cette particularité, contrairement à BASIC, ne présente pas d'intérêt pour l'écriture de programmes ; mais elle peut, comme dans BASIC, obscurcir les raisons d'une erreur à l'exécution : ainsi pour une action à réaliser si le premier caractère du mot affecté au nom "A est égal au premier caractère du mot affecté au nom "B, une erreur de conception peut conduire à une expression telle que SI PREMIER (:A = :B) [action]. Selon la valeur de vérité de :A=:B le message d'erreur sera: F n'est ni vrai ni faux, ou V n'est ni vrai ni faux ; en effet la fonction PREMIER opère de façon valide sur une expression booléenne (puisqu'un booléen est un nom), mais le résultat n'est plus un booléen.

2.1.3 Les entrées-sorties de valeurs booléennes en LOGO et BASIC

En corollaire de cette existence comme "sous-type " d'un type de base, les quantités booléennes peuvent faire l'objet d'opérations d'entrée (lecture au clavier...) ou de sortie (affichage à l'écran), au même titre que les objets du type de base auquel elles appartiennent. Ainsi en BASIC, PRINT (0=0) produit l'affichage du nombre -1. suite à l'instruction INPUT X, si l'utilisateur entre la valeur 0 ou -1, cette valeur pourra être utilisée comme booléen dans le reste du programme. En LOGO, ECRIS (0=0) produit l'affichage du nom VRAI. LISMOT X rendra une valeur utilisable comme booléen dans le reste du programme si l'utilisateur a entré la chaîne VRAI ou la chaîne FAUX.

2.2 Les langages où les booléens constituent un type à part entière

Pour les langages PASCAL, LSE et d'autres, pour le tableur MULTIPLAN et d'autres outils logiciels, les booléens constituent un type à part entière ; c'est-à-dire que les opérations sur les chaînes et les nombres n'opèrent pas sur les quantités booléennes, et que par conséquent les variables doivent faire l'objet d'une déclaration particulière (BOOLEEN en LSE, boolean en PASCAL). En conséquence, les erreurs de type concernant les booléens sont détectées en tant que telles à la compilation.

En contre-partie, les booléens doivent être considérés comme des objets internes, sur lesquels ne peuvent pas porter directement des opérations d'entrée-sortie. En effet, la conception même des organes d'entrée-sortie informatiques impose la communication utilisateur-machine sous forme d'échange de chaînes de caractères ; les chaînes font donc l'objet d'entrées-sorties de façon directe ; les nombres sont entrés et sortis sous le forme de la chaîne de caractères de leur codification en base 10.

2.2.1 L'entrée d'une valeur booléenne

Dans ces langages, la saisie d'une valeur booléenne implique une lecture portant sur une variable de type chaîne ou un nombre, comme dans les lignes de programme PASCAL qui suivent, où la variable ACCORD est supposée déclarée de type booléen, et la variable REPONSE, de type chaîne.

```
writeln ('d'accord ?  (OUI ou NON)' );
readln( REPONSE );
ACCORD:= (REPONSE = 'OUI')
```
[139]

[139] en suivant [JENSEN WIRTH 78]la dernière ligne ci-dessus doit être préférée à l'alternative : IF REPONSE = 'OUI' THEN ACCORD :=true ELSE ACCORD := false

2.2.2 La sortie d'une valeur booléenne

Elle peut être envisagée comme la sortie d'une chaîne de caractères particulière ; ainsi en LSE, `BOOL` étant déclarée comme booléen, `AFFICHER[U] BOOL` provoquera la sortie de la chaîne .VRAI. ou de la chaîne .FAUX. ([LSE 84] page 115) ; en PASCAL, `WRITE(BOOL)` provoquera la sortie de la chaîne `true` ou de la chaîne `false` ([JENSEN WIRTH 78] page 82). Mais ceci doit être envisagé plus comme une facilité pour le programmeur au moment de la mise au point de son programme que comme une procédure normale dans la rédaction d'un programme destiné à un utilisateur ; en effet, un utilisateur doit recevoir des informations sous forme de chaînes ayant un sens pour lui. On préférera donc, pour un programme définitif, l'utilisation d'une alternative, comme dans la ligne de programme PASCAL suivante, où la variable `ACCORD` est supposée déclarée de type booléen :

```
if ACCORD then writeln('OUI') else writeln('NON')
```[140]

A l'intérieur d'un programme, on peut avoir besoin des constantes booléennes, par exemple pour initialiser une variable; on utilise dans ce cas des identificateurs prédéfinis : `true` et `false` en PASCAL, .VRAI. et .FAUX. en LSE.

2.2.3 Le cas des logiciels-outils

Le tableur MULTIPLAN, ou le gestionnaire de fichiers DBASE présentent à la fois un type booléen à part entière, et la possibilité de faire des entrées-sorties sur des valeurs booléennes ; en effet la conception de ces outils fait que toute valeur manipulée doit être accessible à l'utilisateur pour affichage/modification. Les entrées et sorties se font dans DBASE sous forme de suite de caractères ne pouvant être confondues avec des chaînes (`.Y.` et `.N.`) ; dans MULTIPLAN, l'entrée se fait par les fonctions à zéro argument et à valeur constante `VRAI()` et `FAUX()` ; les sorties se font par les chaînes `VRAI` et `FAUX`.

2.3 Le choix d'un langage de programmation

L' enseignant qui veut aborder les booléens avec ses élèves a donc à faire le choix entre des langages (BASIC, LOGO) où les valeurs booléennes sont prises dans les ensembles de valeurs d'un type de base et des langages (PASCAL, LSE) où les booléens constituent un type à part. Les premiers présentent l'avantage de faciliter l'entrée des booléens par l'utilisateur, mais les booléens n'y apparaissent pas comme un type autonome, les résultats d'opérations peuvent dépendre de la codification des booléens comme entiers (BASIC) ou Noms (LOGO), ce qui est générateur de confusions.

Un tableur comme MULTIPLAN, ou un gestionnaire de fichiers comme DBASE présentent l'avantage d'un type booléen à part, et de facilités pour les Entrées/sorties de booléens, mais leur usage est moins général.

3. Des hypothèses sur l'emploi du type booléen dans les premiers apprentissages en informatique

3.1 L'apport de la résolution de problèmes impliquant les booléens

La résolution de problèmes impliquant les booléens doit permettre aux élèves d'acquérir une forme de programmation "booléenne", et donc de simplifier l'écriture d'expressions conditionnelles complexes, et de clarifier le statut des expressions conditionnelles ; en ce sens, elle est un apport aux apprentissages généraux en informatique.

3.2 Les deux axes de problèmes avec leurs difficultés spécifiques

Comme nous l'avons vu ci-dessus, l'emploi du type booléen pour la résolution de problèmes peut s'envisager selon deux axes : le premier concerne la modélisation des objets physiques à deux états, l'autre concerne la modélisation des relations entre objets du monde réel ; cet emploi permet donc aux élèves de nouvelles activités de modélisation, et chacun des axes implique une difficulté nouvelle :

---

[140] De même que ci-dessus, cette forme doit être préférée à la forme : IF ACCORD = true THEN ...

- la modélisation des objets à deux états ne marque pas a priori de rupture par rapport aux activités de modélisation antérieures des élèves. Cependant, la codification par des booléens peut être considérée comme particulièrement *réductrice*: un objet qui a du sens pour l'élève est *réduit* à une information binaire. Une autre difficulté est constituée par le passage d'une forme de programmation utilisant extensivement les alternatives, à une forme plus lisible utilisant le calcul sur les booléens pour la construction de fonctions logiques: la compréhension d'une **condition** comme expression à valeur booléenne, et non plus comme forme verbale proche du langage courant est en jeu dans ce passage. Il pourrait être facilitée si les élèves ont rencontré dans leur études l'utilisation des quantités logiques pour ce type de modélisation (circuits électriques, automatismes...).
- la modélisation des relations entre objets du monde réel suppose pour les élèves, de considérer ces relations comme des objets susceptibles de codification et de calcul ; une autre difficulté est que cette utilisation des booléens trouve sa pleine efficacité seulement avec l'utilisation de types structurés dont les éléments sont des booléens.

3.3 les conséquences du choix d'un langage de programmation

Nous avons indiqué plus haut qu'il nous paraît prématuré, au moment des premiers apprentissages, d'employer un langage d'expression des algorithmes, distinct du langage de programmation utilisé pour le codage en machine ; le choix de ce langage de programmation n'est pas ici indifférent ; en effet, certains langages de programmation (nous avons examiné plus haut le cas de BASIC et LOGO) ne disposent pas d'un type booléen à part entière, et certains effets en retour de l'ordinateur ne s'expliquent que par la codification des booléens comme sous-type d'un type de base. Cette question de codification étant sans doute génératrice de confusion lors d'une première approche, nous pensons judicieux d'aborder les booléens en tant que type, avec un langage de codage disposant d'un type booléen à part entière; comme nous l'avons souligné, les valeurs booléennes sont par conséquent des objets internes, c'est-à-dire qu'on ne peut les faire entrer par l'utilisateur, et que la sortie d'une valeur booléenne se traduit par l'envoi d'une chaîne standard dont la signification peut ne pas être directe pour l'utilisateur. Les élèves seront donc contraints de passer par des nombres ou des chaînes de caractères pour les entrées-sorties, ce qui constituera pour eux une difficulté par rapport aux types déjà connus[141].

---

[141] Nous avons vu également que des logiciels-outils comme MULTIPLAN ou DBASE permettent à la fois l'utilisation d'un type booléen à part entière, et des entrées-sorties de valeurs booléennes ; ces outils permettent une activité de programmation, mais le langage associé présente un caractère particulier, lié aux tâches pour lesquels ils ont été construits; [DAGDILELIS BALACHEFF CAPPONI 91] ont montré à propos de l'itération sur un problème numérique que MULTIPLAN et Pascal consituent pour les élèves des «contextes» différents: dans chacun de ces contextes, on observe des conduites spécifiques liées semble-t'il à des représentations spécifiques, et les méthodes, stratégies ne sont pas réinvesties d'un contexte à l'autre. Concernant l'emploi des booléens dans MULTIPLAN, on trouvera dans [FAVRE-NICOLIN 88], la présentation d'un problème simple de logique résolu à l'aide du tableur Multiplan ; cet article permet d'observer comment sont utilisées les quantités booléennes dans un tableur ; nous pensons cependant que le problème est trop simple pour qu'il justifie l'emploi d'un outil informatique : le tableur se contente d'afficher des valeurs de vérité qui peuvent paraître évidentes, compte tenu des hypothèses.(R. Favre-Nicolin : le tableur et l'option informatique-vers la programmation par objet in Actes du premier colloque francophone sur la didactique de l'informatique)

Chapitre 12 Les booléens: étude expérimentale

choix généraux
des problèmes sur les booléens
résolution par les élèves

Nous avons mené dans une classe d'option informatique des lycées trois séances de recherche de problèmes, dans le but de préciser et de valider certaines des hypothèses du chapitre 11.

1 Choix généraux

1.1 Le type booléen est utilisé pleinement en classe de Première

Nous avons voulu observer une classe de Première utilisant PASCAL comme langage de programmation, pour les deux raisons suivantes :

- compte-tenu de ce que nous avons expliqué ci-dessus, l'observation suppose un langage de codage où les booléens constituent un type à part entière, donc, en pratique, LSE ou PASCAL. Dans la région où nous menons nos observations LSE n'est plus utilisé pour l'option informatique ; en effet, du point de vue des structures de contrôle, et des structures de données, les avantages de LSE sont largement présents dans PASCAL ; comme langage interprété, LSE a pu présenter un avantage au moment où l'on disposait seulement de machines lentes avec peu de mémoire : l'interprétation de chaque ligne de programme au moment de l'exécution utilise peu de place mémoire, et permet de tester et de modifier plus facilement le programme ; l'augmentation de la taille mémoire et de la vitesse des ordinateurs à disposition des élèves permet une compilation facile et rapide dans le cadre d'un "environnement intégré" comprenant un éditeur pleine page : c'est le cas de la variété de PASCAL de marque BORLAND, connue sous le nom de "Turbo-PASCAL", largement utilisée dans les lycées.
  Nous avons donc cherché une classe utilisant PASCAL et nous avons constaté que ce langage est pleinement utilisé seulement en classe de première (c'est-à-dire lors de la deuxième année d'option informatique).
- il nous a semblé que les élèves devaient avoir dominé la structure alternative, et l'emploi des types de base, et connaître les tableaux, en tant que type structuré ; en effet nous avons souligné que certains des problèmes qui nous intéressent supposent l'emploi d'un tableau de booléens.

Pour ces deux raisons, nous avons choisi de mener cette observation dans une classe de Première option informatique.

1.2 Le choix d'une classe de Première restreint la généralité de l'observation

Ce choix présente un inconvénient: comme nous l'avons souligné au chapitre 4, il se produit lors du passage en classe de Première un phénomène d'abandon important (entre 40 et 50 %), et la grande majorité des élèves continuant l'option informatique appartiennent à la filière scientifique (Première S). Le choix d'une classe de Première conduit donc à un public qui n'est pas représentatif de la totalité des élèves s'engageant en seconde dans une alphabétisation informatique; néanmoins, il s'agit ici d'une observation des élèves en situation de résolution de problèmes, et non, comme au chapitre précédent, d'un essai d'ingénierie didactique s'adressant prioritairement aux élèves en difficulté; il nous semble, en conclusion, que cette observation n'aurait pu, pour des raisons institutionnelles, avoir lieu dans une classe représentative de la totalité des élèves s'engageant en seconde dans une alphabétisation informatique, mais que l'observation que nous projetons peut conduire à des résultats valables à condition

de les interpréter relativement au public.[142]

Le choix d'une classe utilisant PASCAL n'implique pas que ce qui suit soit valable seulement dans le cadre de l'utilisation du langage PASCAL; on constatera que le premier problème (INVITATION) suppose seulement l'existence d'un type booléen, et le second de tableaux de booléens. Une résolution est donc possible, même avec un langage ne disposant pas de booléens comme type à part entière ; mais nous pensons que, dans ce cas, les élèves auraient rencontré des difficultés particulières liées à l'existence des booléens comme sous-type d'un type de base ; par contre nous pensons que l'utilisation d'un langage disposant d'un type booléen à part entière comme LSE, ne conduirait pas à des résultats très différents de ceux que nous nous proposons d'observer dans une classe utilisant PASCAL.

2 Les problèmes

Nous avons voulu proposer deux problèmes chacun représentatif d'un des deux axes de modélisation utilisant les booléens :

2.1 INVITATION: un problème portant sur des "objets à deux états" (axe 1)

Dans ce premier problème (INVITATION), les "objets du monde" sont des personnes désignées par leur prénom ; la situation fait que chacune peut prendre deux états : comme il s'agit d'un dîner, elles peuvent être "invitées" ou "non-invitées". De même, le résultat demandé (une "invitation") peut prendre deux états : "possible", ou "impossible" ; compte-tenu des contraintes de l'énoncé, le résultat peut être calculé comme fonction logique des données (l'état de chacune des personnes).

La tâche de l'élève est donc de construire une fonction logique rendant compte des contraintes de l'énoncé ; elle est très proche de la tâche attendue dans des exercices comme par exemple la construction d'un fonction logique rendant compte de l'état "passant" ou "non-passant" d'un circuit électrique constitué d'interrupteurs en série et en parallèle. La situation de résolution est cependant différente en ce sens que les objets en jeu sont plus familiers pour les élèves ; nous avons voulu en effet que les élèves aient un *contrôle sémantique* sur les expressions logiques qu'ils construisent : notre hypothèse est que, par exemple, une écriture comme NON (p ET q) est plus facilement interprétée par des élèves si p et q sont des propositions simples concernant des personnes dans un contexte social, que si elles concernent l'état d'interrupteurs dans un circuit électrique. En contre-partie, la codification est particulièrement réductrice, puisqu'ici ce sont des personnes (donc des "objets" ayant une valeur évocatrice forte), mais ce caractère réducteur fait partie de la "distance" entre la situation familière et les données codifiées, dont nous attendons un effet sur les représentations des élèves. INVITATION est donc conforme aux spécification des problèmes que nous avons énoncées au chapitre **3**.

Nous pensons également que dans une situation familière les élèves envisagent spontanément une résolution s'appuyant sur des structures alternatives; mais ces alternatives, dès que les clauses deviennent complexes, sont difficilement articulables et validables. Les élève doivent donc, dans un second temps, considérer **l'emploi des booléens**, et le **calcul sur les booléens** comme un moyen d'écriture d'un programme permettant une validation plus facile, et plus de lisibilité (hypothèse 3.1, chapitre précédent).

---

[142] Les dernières décisions ministérielles (avril 91) sur l'organisation des lycées conduiront semble-t'il à ce que l'option informatique ne soit plus proposée aux élèves de seconde indifférenciée, mais seulement aux élèves des classes de première et terminale scientifique. Le choix, dicté par les circonstances, d'une observation en classe de première pour cette partie de notre thèse, est donc d'une actualité particulière: si nous vérifions que la constitution de systèmes de représentation différenciés et hiérarchisé se pose également en classe de première scientifique comme une étape importante vers les acquisitions en informatique, nous aurons également montré que la démarche dans laquelle nous nous inscrivons a une validité au delà des décisions circonstancielles des autorités ministérielles.

Enoncé du premier problème (INVITATION)

J'ai 5 amis : Marie, Marc, Luc, Janine et Jean.

Je souhaite les inviter à dîner, mais il y a des incompatibilités d'humeur et des préférences, que je traduis par les "clauses" suivantes :

- clause 1 : "Marie et Jean ne s'entendent pas : il ne faut pas les inviter ensemble ".

- clause 2 : "Marc et Marie ne viendront pas l'un sans l'autre : si j'invite l'un , il faut que j'invite l'autre".

- clause 3 : "Si j'invite Janine, il faut que j'invite Luc ou Jean, mais on ne peut pas les inviter tous les trois ensemble"

On veut écrire un programme qui permet de savoir si une invitation (c'est-à-dire un groupe d'amis à inviter) est possible :

Ecris un programme qui pose 5 questions "On invite Marie (O/N)" ... puis calcule et affiche la valeur de vérité de chaque clause (c'est-à-dire si la clause est ou non respectée).

Une solution utilisant les booléens :

(nous codons cette solution en PASCAL, langage utilisé par les élèves ; PASCAL ne présente pas ici de particularités qui diminueraient le caractère de généralité de cette solution).

```
program invitation ;
var Marie , Janine, Jean , Marc , Luc ,
    clause1, clause2, clause3, possible : boolean    ;
    rep:char ;

begin
write('Marie  ')  ; readln(rep) ; Marie :=(rep='o') ;
write( 'Janine ') ; readln(rep) ; Janine:=(rep='o') ;
write('Jean ')    ; readln(rep) ; Jean  :=(rep='o') ;
write( 'Marc ')   ; readln(rep) ; Marc  :=(rep='o') ;
write('Luc   ')   ; readln(rep) ; Luc   :=(rep='o') ;

clause1:= not (Marie and Jean)
clause2:= (Marie = Marc)
clause3:=(Luc or Jean or not Janine) and not (Luc and Jean and Janine)
writeln('clause1  ',clause1 ) ;
writeln('clause2  ',clause2 ) ;
writeln('clause3  ',clause3 ) ;
possible:= clause1 and clause2 clause3 ;
writeln('invitation possible : ' , possible)
end.
```

Notes et variantes:

Nous avons voulu que les trois clauses présentent une difficulté croissante dans la construction d'une fonction logique:

- le calcul de la valeur de vérité de la première clause se déduit directement de l'énoncé.
- la seconde clause s'exprime de cette façon particulièrement concise, si l'on traduit l'énoncé comme l'équivalence logique des propositions `[Marc est invité]` et `[Marie est invitée]`, et si l'on reconnait que l'égalité des booléens traduit l'équivalence logique des propositions.
- la troisième clause pourrait se traduire directement à l'aide de l'implication logique et du `OU exclusif` (XOR) par: `Janine implique (Luc XOR Jean)`
  Le connecteur "implique" n'existe généralement pas comme opérateur primitif dans les langages de programmation. [JENSEN WIRTH 78] indique que, comme `false` est avant `true`, dans l'ordre sur les booléens, l'implication : `p implique q`, peut s'écrire `p <= q` (`p` inférieur ou égal à `q`). Mais ceci constitue une particularité de PASCAL non généralisable, et suppose connue la table de vérité de l'implication. C'est pourquoi nous avons choisi dans cette solution pour la clause 3 une expression que

les élèves peuvent construire, et qui utilise l'équivalence entre les propositions (P implique Q), et (Q ou non P). Le connecteur XOR peut exister dans le langage, mais n'est certainement pas connu des élèves.

- le calcul de clause3 peut également s'exprimer sous forme disjonctive par exemple
  clause3:=(Jean and not Luc) or (Luc and not Jean) or not Janine
  Cette écriture peut se déduire de la précédente par l'emploi de théorèmes logiques; elle peut également se déduire de l'observation de la table de vérité suivante:

Jean	Luc	Janine	clause 3
V	V	V	F
V	V	F	V
V	F	V	V
V	F	F	V
F	V	V	V
F	V	F	V
F	F	V	F
F	F	F	V

  Il est peu probable que les élèves connaissent les théorèmes permettant d'obtenir cette forme (non (P et Q) = non P ou non Q ; distributivité de et sur ou ; (P et Q) ou P = P). Ils ne connaissent sans doute pas non plus les tables de vérité, mais nous faisons l'hypothèse qu'ils peuvent construire une forme disjonctive en raisonnant sur les situations respectant la clause.
- cette clause étant plus complexe, il peut être intéressant de décomposer son calcul , en utilisant l'affectation à des variables intermédiaires :

```
clause3a:= (Luc or Jean or not Janine)
clause3b:= not (Luc and Jean and Janine)
clause3:= clause3a and clause3b.
```

- la sortie des résultats se fait par des instructions d'affichage direct d'un booléen ; on obtiendra donc par exemple à l'utilisation :

```
clause1   true
clause2   true
clause1   false
invitation possible : false
```

  ce qui est suffisant pour une première version du programme. Une version définitive utiliserait une alternative pour chaque sortie ; par exemple :

```
write('clause1  ') ;if not clause1 then write('non ') ;
writeln('vérifiée')
```

- le calcul d'une valeur de vérité pour la possibilité d'une invitation (variable booléenne possible) n'est pas demandé de façon explicite dans l'énoncé ; il constitue cependant le but de la résolution. Nous avons indiqué dans les chapitres précédents qu'un calcul conduisant à l'affectation à une variable du résultat nous semble plus valable qu'un simple affichage. Dans bien des problèmes posés en initiation, les deux démarches sont possibles, mais seule la première permet que le résultat soit réutilisé dans un autre module éventuel. Ici, le calcul de la valeur de la variable possible permet de valoriser le calcul des valeurs de vérité des clauses et l'affectation de ces valeurs à des variables booléennes.

Quelles conduites des élèves attendre devant ce problème :

On attend que dans ce problème, les élèves reconnaissent assez facilement le type "booléen" comme convenant aux données du problème, mais rencontrent des difficultés pour l'entrée des données par l'utilisateur, ainsi que pour l'expression des fonctions booléennes rendant compte de la valeur de vérité des clauses. Comme indiqué plus haut, la démarche spontanée sera la construction des fonctions booléennes à l'aide d'alternatives (forme "alternatives complètes" ou "remplacement conditionnel" exposées au chapitre 11). Cette démarche pourra réussir pour le calcul des 2 premières clauses, mais se heurtera à des problèmes de complexité dans le calcul de clause3. L'intervention de l'enseignant sera nécessaire pour diriger les élèves vers un calcul sur des booléens (forme "booléene" de programmation).

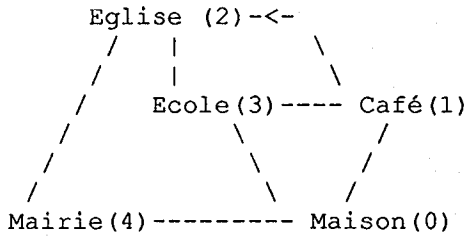
2.2 VILLAGE: un problème où les booléens codifient des relations entre objets du monde (axe 2)

Dans le second problème (VILLAGE), les booléens doivent servir à exprimer des relations entre objets du monde ; il s'agit en fait d'un problème sur les graphes orientés, mais comme les élèves ne connaissent pas de méthode générale de codification des graphes, leur première difficulté sera de trouver une codification des relations entre les objets en jeu dans le problème; nous avons cependant cherché un problème où intervient seulement la lecture d'un tableau de booléens, et non la multiplication de matrices booléennes que nous avons utilisée dans l'exemple du chapitre précédent (§1.3.3).

D'autre part, nous avons pris soin de proposer un graphe non symétrique, où l'idée d'associer un booléen à un couple s'impose plus facilement: `(a,b)` peut être `Vrai`, et `(b,a)` `Faux`.

Enoncé du second problème (VILLAGE)

Ma maison est dans un village qui comprend également un café, une église, une école et une mairie ; il y a des rues ; voici le plan :



La rue du Café à l'Eglise est en sens unique; (comme je circule à bicyclette, il ne m'est pas possible de l'emprunter de l'église vers le Café). Le dimanche, je fais une promenade ; je part de ma maison et j'y reviens après être passé une seule fois en chacun des autres bâtiments marqués sur le plan ; pour m'en rappeler, je note dans l'ordre les points où je suis passé ; par exemple : 1 3 2 4.

Je voudrais un programme qui m'indique si une promenade est possible ; par exemple : 1 2 3 4 , ou 2 1 3 4 ne sont pas possibles.

Ecris un programme où l'utilisateur entre une suite de 4 chiffres désignant une promenade, et où l'ordinateur répond "possible" ou "impossible".

Une solution :

Elle utilise la codification classique des graphes orientés comme tableau à double entrée indexé sur les noeuds du graphe, et à valeur booléenne ; ce tableau étant constitué principalement d'éléments ayant la valeur fausse, il est plus concis de donner dans une double boucle cette valeur à tous les éléments, puis de donner la valeur vraie à tous les éléments correspondant à un arc du graphe.

```
program village;
type t_batiment=0..4;
var tableau_chemin:array[t_batiment,t_batiment] of boolean ;
    liste_etapes : array[0..5] of t_batiment;
    i:0..5;
    present, suivant :t_batiment;

procedure init;
var i,j:t_batiment;
begin
    for i:=0 to 4 do
        for j:=0 to 4 do  tableau_chemin[i,j]:=false;
    tableau_chemin[1,0]:=true;
    tableau_chemin[0,1]:=true;
    tableau_chemin[3,0]:=true;
    tableau_chemin[0,3]:=true;
    tableau_chemin[0,4]:=true;
    tableau_chemin[4,0]:=true;
```

```
    tableau_chemin[1,2]:=true;
    tableau_chemin[2,3]:=true;
    tableau_chemin[1,3]:=true;
    tableau_chemin[3,1]:=true;
    tableau_chemin[3,2]:=true;
    tableau_chemin[2,4]:=true;
    tableau_chemin[4,2]:=true;
end; {init}

begin {programme principal}
    init;
    {saisie du parcours}
    for i:=1 to 4 do
 begin
      write ('étape n° ',i, '? ');
      readln(rep);
      liste_etapes[i]:=rep
 end;
    {calcul du booléen possible}
    possible:=true;
    liste_etapes[0]:=0;
    liste_etapes[5]:=0;
    for i:=0 to 3 do
 begin
      present:=liste_etapes[i];
      suivant:=liste_etapes[i+1];
      possible:=possible and tableau_chemin[present,suivant]
      end;
    writeln(possible)
end.
```

Notes et variantes :

Il est possible de "sortir de l'itération" dès qu'un "trajet élémentaire impossible" a été rencontré, et d'employer le théorème logique `V AND P = P`; on obtient une itération plus concise, mais avec une condition de sortie plus complexe

```
 while possible and i<4 do  begin
                present:=liste_etapes[i];
                suivant:=liste_etapes[i+1];
                possible:= tableau_chemin[present,suivant]
                i:=i+1
                                end
```

La boucle de calcul du booléen possible est généralisable à un graphe ayant un nombre quelconque de sommets ; ici, cependant, le nombre de sommets est suffisemment petit pour éviter le vecteur `liste_etapes`, la boucle d'entrée des étapes, et la boucle de calcul du booléen `possible`. La solution attendue d'un élève est donc plutôt:

```
write ('étape n° 1 ');
readln(etape1);
write ('étape n° 2 ');
readln(etape2);
write ('étape n° 3 ');
readln(etape3);
write ('étape n° 4 ');
readln(etape4);
possible:= tableau_chemin[0,etape1] and
tableau_chemin[etape1,etape2] and tableau_chemin[etape2,etape3]
and tableau_chemin[etape3,etape4] and tableau_chemin[etape4,0]
```

Qu'attend t'on du problème ?

Comme vu plus haut, la première difficulté sera de concevoir qu'il faut une codification du plan du village utilisable pour l'ordinateur. les élèves devront ensuite tenter d'élaborer une telle codification, puis construire la fonction logique résultat. Le nombre de sommets du graphe est limité à 5 pour qu'un calcul non-itératif du résultat soit possible, comme dans la variante ci-dessus ; mais le graphe est suffisamment complexe pour que la résolution par alternatives ne soit pas envisageable, et donc pour

que le problème pose la question de sa codification.

2.3 Conditions de résolution

Présentation de la classe :

Cette classe de Première Option informatique du lycée ALLENDE d'HEROUVILLE comprend 9 élèves dont 7 de 1ère S (Première Scientifique) et 2 de 1ère B (Première Sciences Economiques). Le type booléen a été introduit notamment avec la codification des états d'un circuit électrique. Un circuit étant donné, comportant un générateur, électrique, des interrupteurs et une lampe, écrire un programme qui, pour la donnée d'un état des interrupteurs, donne pour résultat l'état de la lampe : l'étude s'est limitée en fait aux circuits les plus simples : deux interrupteurs en série (connecteur ET), d'une part, deux interrupteurs en parallèle d'autre part (connecteur OU). Le professeur a noté une réticence des élèves à employer le type booléen due, selon elle, à l'impossibilité d'entrer une donnée de type booléen au clavier. Les trois séances qui ont été consacrées à la résolution des problèmes présentés au paragraphe précédent s'ajoutent en fait à une progression qui ne prévoyait plus de travail particulier sur les booléens.

Première séance : une heure (13-12-88)

Au cours de la première séance, nous avons demandé aux élèves d'effectuer hors de la présence des ordinateurs, le travail sur papier préparatoire au passage sur machine Le résultat de cette recherche a été ramassé en fin de séance, mais le travail demandé n'a pas eu le caractère d'une épreuve d'évaluation ; nous avons cherché à ce que l'analyse soit produite de façon spontanée, sans que des contraintes d'expression interviennent.

Pour chaque problème, le compte-rendu de cette séance est constitué par un bref rapport sur les conditions dans lesquelles s'est déroulée la recherche des élèves, puis par une analyse de 7 protocoles constitués des traces écrites laissées par les élèves suite à leur recherche ; cette analyse est suivie d'un tableau récapitulatif commenté.

Seconde et troisième séance : 1h30 (15-12-88) 1h (19-12-88)

La seconde et la troisième séance ont été consacrées au travail sur machine pour la résolution de chacun des problèmes ; à chacune de ces séances, nous avons conservé les programmes produits par les élèves et enregistré certains élèves en situation de travail guidé. Le premier travail des élèves au cours de ces deux séances, a consisté à coder en machine l'analyse produite pour chaque problème à la première séance. Nous avons ensuite demandé aux élèves de chercher des solutions utilisant davantage la structure booléenne que les productions spontanées. Pour les aider, nous leur avons proposé une structuration des données sous forme d'un programme à compléter pour chaque problème. Le compte-rendu de ces séances comporte pour chaque problème l'analyse des programmes laissés par les élèves, et le dépouillement commenté des enregistrements (même remarque que pour la résolution sur papier).

3 La recherche du premier problème (INVITATION)

3.1 La recherche sur papier

3.1.1 compte-rendu de l'observation

Elle occupe environ 35mn ; les élèves reconnaissent spontanément un problème où "*il va falloir utiliser les booléens*". Dans un premier temps, les élèves se livrent à une recherche "à la main" des invitations possibles, ce à quoi ils réussissent en général.

Les élèves auraient pu, à partir de cette recherche tenter l'écriture d'un programme, où l'ordinateur produirait sa réponse, non par un calcul de la valeur de vérité des clauses, mais par une simple confrontation des données aux "invitations possibles" déterminées "à la main". Ce programme n'aurait pas été tout à fait conforme à l'énoncé, puisque celui-ci demande la valeur de vérité de chaque clause. Les élèves ne tentent pas l'écriture d'un tel programme.

Nous les orientons alors vers le calcul de la valeur de vérité de chaque clause; on trouvera le résultat de cette recherche dans les protocoles ci-dessous.

3.1.2 Analyse des productions écrites des élèves pour le problème 1 (Tableau récapitulatif)

Les protocoles sont constitués des traces sur papier laissés par les élèves, individuellement ou en groupe. On trouvera en annexe 11 une description de chaque protocole. Nous dressons ici, un tableau récapitulatif. Pour chaque protocole, nous distinguons:

- la forme de l'analyse : il peut s'agir de l'expression d'un procédé de calcul en langage naturel (que nous appelerons algorithme) ou d'un programme PASCAL plus ou moins complet,
- le type employé pour représenter les quantités logiques, suivi de la mention "(implicite)" si la déclaration n'est pas faite.
- la façon dont la question de l'entrée des valeurs de vérité de chaque invitation a été résolue ; il peut s'agir d'une entrée directe, si le type choisi pour représenter ces données le permet, où de l'entrée d'une chaîne de caractères suivie d'une affectation conditionnelle,
- les clauses dont la programmation a été abordée, avec la mention "correct" s'il n'y a pas d'erreur, "fausse" sinon,
- les connecteurs logique employés,
- la forme de programmation: nous avons distingué en 1.3.3, trois formes de programmation pour cet axe de problème:
    - l'alternative complète où une variable logique reçoit une valeur dans les deux branches de l'alternative.
    - le "remplacement conditionnel" où une variable logique reçoit une valeur initiale Vraie , puis est éventuellement modifiée par une suite d'alternatives ne comportant pas de partie "SINON"; cette forme permet de réaliser le OU logique par deux conditions successives, et simplifie l'écriture des alternatives en chaîne qui permettent de réaliser le ET logique.
    - la forme "booléenne" qui exprime, sans alternatives, le calcul d'une valeur logique résultat à l'aide des connecteurs logiques.

Aucun protocole n'emploie une forme de programmation qui pourrait être classée avec la forme "booléenne". La classification se complique par le fait que les élèves emploient dans certains protocoles, l'affichage d'un message au lieu de l'affectation à une variable résultat. Ceci se rencontre particulièrement avec des alternatives complètes: dans ce cas les deux branches de l'alternatives sont constituées d'un l'affichage portant à la connaissance de l'utilisateur la valeur de vérité de la clause concernée (SI <condition> ALORS AFFICHER 'Vérifié' SINON AFFICHER 'Non vérifié'). Nous distinguerons donc la forme "alternative-complète affectation" de la forme "alternative-complète affichage". L'affichage d'un message se rencontre également avec des alternatives sans partie SINON; dans ce cas, l'affichage d'un message se produit seulement quand la clause n'est pas respectée: nous appelerons cette forme "affichage conditionnel". Elle présente pour les élèves les avantages de la forme "remplacement conditionnel", tout en ne demandant pas la maîtrise de l'affectation. Un protocole emploie des alternatives sans partie SINON, mais le statut de l'action reste dans le flou: nous noterons donc: "(??)conditionnel".

protocole	forme de l'analyse	type des données	entrée des données	clauses programmées	connecteurs employés	forme de programmation
A 1	programme	numérique (implicite)	via chaîne	1 (correct) 2 (faux)	and	alternative-complète affichage
A 2	amorce algorithme.	booléens tableau (implicite)	non résolue			
A 3	algorithme	booléen dans l'algorithme, numérique dans le programme	directe (manque affectation si succès)	1,2,3		alternatives en chaîne, remplacement conditionnel
A 4	programme	numérique	directe	1,2(correct) 3 (faux)	and, différent	remplacement conditionnel
A 5	paraphrase de l'énoncé	booléens	non résolue			
A 6	programme	booléens (implicite)	non résolue	1 (confus)	and	affichage conditionnel
A 7	programme partiel	numérique (implicite)	via une chaîne	1,2(correct) 3 (faux)	and + différent	(??) conditionnel

Analyse du tableau :

- les variables sont déclarées de type booléen seulement par ceux qui produisent une analyse assez éloignée d'un programme effectif.
  On observe d'ailleurs, avec le protocole 3, un passage des booléens (dans l'algorithme) aux entiers (dans le programme) ; ceci reflète nettement que les élèves choisissent d'abord le type des données en fonction de leur nature dans le problème ; puis dans l'écriture d'un algorithme plus précis ou d'un programme ils rencontrent des difficultés avec ce type qui ne leur est pas familier, et ils sont conduits à remettre en cause ce choix, au profit d'un type qui leur soit mieux connu.
- Les difficultés qui conduisent à écarter le type booléen sont de deux types :
  - l'entrée des données, qui ne peut être faite de façon directe pour les booléens ; cette difficulté est résolue dans deux protocoles, qui écartent néanmoins le type booléen,
  - l'écriture des conditions à l'aide de connecteurs logiques.
    Le connecteur `and` est employé dans 4 protocoles sur 7 ; les élèves l'ont employé auparavant pour écrire la partie condition d'alternatives. Dans un cas, cependant, l'emploi d'alternatives en chaîne dispense du `and`. Les élèves n'emploient ni `or` ni `not`; en effet la forme "affichage conditionnel" permet d'éviter le recours au `or`, et il suffit de "renverser une alternative" pour éviter le recours à la négation d'une condition. Dans deux protocoles, on trouve l'emploi de l'opérateur "différent de" (qui s'écrit **<>** en PASCAL) ; il a la propriété d'opérer aussi bien sur les booléens que sur les autres types de base : comme connecteur logique il a la valeur de "non-équivalent", et comme opérateur sur des quantités numériques, les élèves l'ont utilisé certainement à maintes reprises. Un élève utilise l'addition pour (re)construire le `and` logique: sa construction est correcte, mais traduit une défiance vis-à-vis des booléens. Un autre élève utilise systématiquement des alternatives en chaîne, plutôt que le connecteur `and`.
- La forme de programmation employée par les élèves: sauf dans 2 protocoles, le programme est envisagé comme une suite d'affichage plutôt que comme le calcul de valeurs logiques; à une exception près (protocole A4), les élèves n'ont pas non plus envisagé de donner en fin de programme une valeur de vérité "possible" ou "impossible" tenant compte de l'ensemble des clauses. En dehors de cette perspective de réutilisation des valeurs de vérité trouvées pour chacune des clauses, il n'y a

pas, en effet de motivation particulière à conserver dans une variable (booléenne) ces valeurs, et dans cette interprétation de l'énoncé, les formes "alternatives affichage", et "affichage conditionnels" suffisent. L'énoncé aurait pu être plus précis sur ce point.
Les élèves qui vont le plus loin dans l'analyse sont ceux qui emploient la forme "remplacement conditionnel", ou "affichage conditionnel"; en effet, l'alternative complète se révèle non adaptée à partir du moment où les conditions sont composées, et où les élèves n'utilisent pas les connecteurs logiques pour l'écriture des conditions. Dans un cas (A3), un élève emploie la forme remplacement conditionnel, mais omet l'affectation préalable de la variable résultat.

3.2 Le travail sur machine

3.2.1 Une version *spontanée* et une version *booléenne*

Les deux séances suivantes (respectivement de 1h30 et de 1h) ont été consacrées au travail sur machine ; dans un premier temps, nous avons demandé aux élèves d'écrire le programme correspondant à l'analyse faite à la première séance (ce que nous appelons dans la suite la version *spontanée*), et de l'exécuter ; nous leur avons montré ensuite que l'emploi de variables de type numérique pour représenter des quantités logiques, et l'emploi d'alternatives rend leur programme difficile à valider ; nous les avons donc convié à utiliser le type booléen et des structures non alternatives ; de façon à leur simplifier cette démarche, nous leur avons fourni un programme à compléter où les déclarations de type, et l'entrée des données sont résolues :

```
program invitation ;
{complétez ce programme pour résoudre le problème 1 ;
celui qui emploie le moins de "IF...THEN " a gagné !
vous pouvez bien sûr employer d'autres variables,
à condition de les déclarer.  }
var Marie , Janine, Jean , Marc , Luc ,
 clause1, clause2, clause3, clause4, clause5 : boolean ;
     rep:char ;
begin
write('Marie  ')  ; readln(rep) ; Marie :=(rep='o') ;
write( 'Janine ') ; readln(rep) ; Janine:=(rep='o') ;
write('Jean ')    ; readln(rep) ; Jean  :=(rep='o') ;
write( 'Marc ')   ; readln(rep) ; Marc  :=(rep='o') ;
write('Luc   ')   ; readln(rep) ; Luc   :=(rep='o') ;

    clause1:=
    clause2:=
    clause3:=
     writeln('clause1  ',clause1 ) ;
     writeln('clause2  ',clause2 ) ;
     writeln('clause3  ',clause3 ) ;
end.
```

Les programmes obtenus en complétant cette amorce de programme constituent la version *booléenne*.

Les programmes produits par les élèves dans les deux versions ont été tirés sur imprimante et conservés ; pour un élève d'une part, et un groupe de deux élèves d'autre part, la recherche guidée d'une solution utilisant les booléens a donné lieu à un entretien enregistré sur magnétophone. En annexe **11** (§ 4), nous donnons les deux versions données par chacun des élèves (ou groupes d'élèves), et une brève analyse de ces versions. Compte-tenu d'absences à l'une des séances, certains élèves ont rendu une seule des deux versions.

3.2.2 Du travail sur papier au programme "version spontanée"

Nous établissons ci-dessous un tableau récapitulant les évolutions des productions sur papier (analysée ci-dessus) à la version spontanée (codage en machine de ces productions). Nous signalons une évolution éventuelle concernant le type des données, et la forme de programmation.

Tableau récapitulatif comparaison entre les productions sur papier, et la version spontanée sur ordinateur	
Jérôme protocoles A3 et B3	pas d'évolution (les détails erronés sont corrigés, et le programme est correct)
Hervé protocoles A4 et B4	pas d'évolution : l'erreur sur clause3 est corrigée.
Denis protocoles A5 et B5	passage des booléens aux entiers pour la codification des quantités logiques ; résolution erronée.
Alain protocoles A7 et B7	persistance de l'utilisation des entiers pour la codification des quantités logiques ; persistance de l'erreur dans la clause3

Commentaire

La seule évolution notable est celle de Denis qui avait produit sur papier seulement une paraphrase de l'énoncé, où cependant le type booléen était choisi pour les quantités logiques. Confronté à l'écriture d'un programme, il fait cette fois, comme les autres élèves lors du travail sur papier, le choix d'un type numérique. Dans deux cas sur quatre, le programme reste erroné, malgré la facilité que pourrait apporter la présence de l'ordinateur pour faire des vérifications. On peut penser qu'il s'agit d'une conséquence du manque de lisibilité de programmes employant abondamment l'alternative.

3.2.3. Le calcul des clauses dans la version booléenne (La version booléenne est obtenue suite à l'indication donnée dans le programme à compléter)

1. CLAUSE 1

protocole	calcul
B2	`(Marie and (not Jean)) or (Jean and (not Marie)) or ((not Marie) and (not Jean))`
B3	`not( n[1] and n[5])` *(pour : not(marie and jean))*
B6	`not( Marie and Jean)`
B7	`not (invitation[1] and invitation[5])` *(pour : not(marie and jean))*

Dans trois protocoles sur 4, le calcul est traduit directement de l'énoncé. Ces élèves n'ont pas tenté de transformer l'expression pour "distribuer" la négation. Nous avons pu vérifier à l'aide des enregistrements, que la tautologie

`NON (P et Q) équivalent à (NON P) ou (NON Q)`

n'est pas connue des élèves. Les élèves du protocole B2 ont voulu, plutôt que de traduire l'énoncé (en employant la négation), énoncer tous les cas où l'invitation est possible et ont donc obtenu la forme canonique disjonctive.

2. CLAUSE 2

protocole	calcul
B2	`(Marie and Marc) or ((not Marie) and (not Marc))`
B3	`(n[1] = n[2])` *(pour Marie=Marc)*
B6	`Marc = Marie.`
B7	`not(invitation[1]<>invitation[2])` *(pour not(Marie différent de Marc))*

La condition `invitation[1]<>invitation[2]` avait été trouvée par Alain (protocoles A7 et B7) dès la recherche sur papier ; elle impliquait alors des entiers, et n'était pas précédée d'une négation à cause de la forme de l'alternative dont elle était la partie condition ; l'élève ne tente pas de la simplifier. On verra dans l'enregistrement d'un entretien que l'expression du protocole B3 est obtenue suite aux interventions de l'enseignant. On retrouve dans la réponse du protocole B2 la forme canonique disjonctive.

3. CLAUSE 3

protocole	caicul

B2	c31 := Janine and Jean;
	c32 := Janine and Luc;
	c33 := (not Janine) and (not Luc) and (not Jean);
	clause3:=c31 or c32 or c33;

B3	cc1:=not(n[4] and n[5] and n[3]);
	(pour not *(Janine* and *Luc* and *Jean)*)
	cc2:=not(n[4] and (not n[5] and (not n[3])));
	(pour not *(Janine* and *not Luc* and *not Jean)*)
	cc:=cc1 and cc2 ;

B7	not (invitation[4] and (invitation[3]=invitation[5])); *(pour* not *(Janine* and *(Luc = Jean)*))

Les élèves des protocoles B2 et B3 ont décomposé le calcul à l'aide de l'affectation à des variables intermédiaires suite à nos indications (voir le dépouillement des enregistrements).

Pour les élèves du protocole B2, cette décomposition sert une fois de plus à exprimer le résultat comme disjonction des cas où l'invitation est possible ; mais ici, le cas où Janine, Jean et Luc sont présents simultanément donne la valeur Vraie à Clause3, en contradiction avec l'énoncé. On peut penser que, pour ces élèves, la disjonction `c31 or c32` est comprise comme exclusive. Egalement, le cas où Jean et Luc sont présents sans Janine n'est pas autorisé. On verra dans le compte-rendu de l'entretien avec ces élèves que le manque de temps a abrégé la résolution de cette partie. Cette clause, mettant en jeu trois propositions, conduit à 8 propositions élémentaires, alors que les précédentes mettaient en jeu deux propositions seulement, et donc 4 propositions élémentaires. Ici, les élèves tentent des formulations disjonctives non canoniques, et tombent dans les pièges de telle formulations: ambiguïté du OU, oubli de cas ... Les élèves peuvent ils progresser davantage sans des acquisitions en logique portant par exemple sur les tables de vérité?

Pour Jérôme (protocole B3), la décomposition épouse la formulation de l'énoncé : "on ne peut pas les inviter tous les trois ensemble" est traduit par : `cc1:=not(n[4] and n[5] and n[3])`; "Si j'invite Janine, il faut que j'invite Luc ou Jean" est traduit par : `cc2:=not(n[4] and (not n[5] and (not n[3]))`.

Alain (protocole B7), par contre, n'utilise pas de décomposition, et obtient une formulation particulièrement élégante, qui doit certainement plus au raisonnement qu'aux transformations formelles.

3.3 Entretiens sur la résolution du premier problème (INVITATION)

Les entretiens portent sur la rédaction de la version booléenne du programme; les comptes-rendus sont en annexe 11 (§5 et 6). Ils permettent de préciser les observations faites au paragraphe précédent sur les réponses des élèves pour cette version booléenne , et mettent en évidence la façon dont cette version a été obtenue.

3.3.1. Les difficultés à utiliser les booléens (analyse de l'entretien avec Jérôme annexe 11 §5)

Le calcul de la clause 1 et de la clause 2 est obtenu suite à des interventions importantes de l'observateur. L'élève accepte que les variables soient de type booléen, mais pense la résolution en terme d'alternatives, y compris jusqu'à l'écriture de la clause 2. Ensuite, le principe de l'affectation à un booléen est acquis par l'élève, mais il rencontre de nombreuses difficultés dans l'emploi des connecteurs : distributivité de NOT sur AND, passage d'une forme négative à une forme affirmative. Cette difficulté à utiliser les connecteurs logiques est à mettre en rapport avec l'utilisation systématique par cet élève du remplacement conditionnel et d'alternatives en chaîne dans la version spontanée, précisément pour éviter le recours aux connecteurs logiques.

3.3.2. Signification de l'affectation et des calculs sur les booléens (analyse de l'entretien avec Jean-Manuel et Valérie annexe 11 §5)

Des "conditions" aux booléens.

Une première difficulté au cours de l'entretien est le passage du point de vue "condition" au point de vue "valeurs booléennes": pour Jean-Manuel, au début, et pour Valérie jusqu'à la fin, une expression logique n'a de sens que comme premier argument d'une alternative. De même, les connecteurs doivent, dans cette conception, relier des "conditions" qui elle-même sont écrites à l'aide d'opérateurs de comparaison; c'est l'explication de formulations proposées par les élèves telles que: `marie = Vrai OR jean=Faux` pour `marie OR NOT jean`.

La compréhension de l'affectation d'une valeur booléenne:

La compréhension acquise sur d'autres types, n'est pas transférée : Jean-Manuel interprète l'affectation `marie:=(rep='o')` " comme «`marie` *correspond à* `(rep='o')`». Il lui faut donc une interprétation particulière valable pour les booléens. Cette attitude de Jean-Manuel rappelle la compréhension de l'affectation chez les débutants que nous avons observée lors de la résolution de problèmes sur les chaînes de caractères: une forme verbale impliquant une vague correspondance entre les objets. Bien sûr, cette représentation est liée à la compréhension des expressions logiques comme "conditions"; la notion même de valeur logique n'étant pas acquise, il n'est pas étonnant que le transfert d'une valeur logique par l'affectation ne le soit pas non plus. De même, l'utilisation de l'affectation pour décomposer un calcul n'est pas acquise de façon immédiate; les élèves tentent d'ailleurs pour ce faire une curieuse affectation en chaîne: `Clause3:= C31:=Janine...`

L'emploi des connecteurs propositionnels.

Jean-Manuel pense systématiquement les expressions booléennes sous forme d'une disjonction des cas possibles ; il est conduit à utiliser `AND` , `NOT` et `OR`. Dans le calcul de CLAUSE1 et CLAUSE2, les élèves produisent la forme canonique disjonctive. Pour ces clause, le nombre de fonctions logiques élémentaires est limité à 4, ce qui rend plus aisé l'écriture de cette forme. Néanmoins, l'oubli d'un cas dans la première expression produite pour CLAUSE1 montre un manque de familiarité avec le connecteur disjonctif `OR`. Pour le calcul de CLAUSE3, le nombre de fonctions logiques élémentaires est de huit; les élèves tentent une forme disjonctive non canonique, qui laisse subsister deux erreurs.

3.4 Conclusion sur le premier problème (INVITATION)

3.4.1 Dans leur démarche spontanée, les élèves éliminent le type booléen au profit des types antérieurs, et le calcul sur les booléens au profit des alternatives

- les élèves reconnaissent en début de recherche qu'il s'agit de traiter des quantités logiques, et déclarent comme booléens les variables représentatives de ces quantités, mais au fur et à mesure qu'ils avancent dans la résolution, ils abandonnent le type booléen pour adopter le type entier (voir tableaux § 5.1 et 5.2).
- au cours du travail sur papier, comme au cours de la programmation de la version spontanée, l'ensemble des élèves envisage la programmation seulement sous forme d'alternatives ; ainsi, l'emploi des connecteurs logiques est limité au minimum indispensable, et il n'y a pas d'affectation à une variable booléenne.

3.4.2 La première difficulté est l'entrée des valeurs booléennes. Ensuite les difficultés peuvent être liées au manque de pratique du calcul sur les booléens et à leur représentation des expressions logiques

- le fait que le type booléen est écarté peut s'expliquer par l'impossibilité d'entrer des valeurs booléennes au clavier ; on remarque cependant que deux groupes résolvent la question de l'entrée de valeurs logiques sous la forme de l'entrée d'une chaîne, suivie d'une alternative ; cette solution reste valable si la valeur logique est affectée à une variable booléenne.
- de façon générale, les élèves ne sont pas à l'aise avec les booléens : difficulté à

interpréter l'affectation, difficulté à employer les connecteurs logiques; ceux qui ont acquis la forme "remplacement conditionnel" disposent ainsi d'un outil leur permettant de programmer avec succès. en évitant le recours au calcul sur les booléens, et en particulier l'emploi du NOT et du OR: OR n'est pas utilisé, sauf pour exprimer une disjonction des cas; dans cet emploi, en effet, l'ambiguïté sur le caractère exclusif de ce connecteur, et la question des priorités entre connecteurs peuvent ne pas se poser.

- le connecteur "=" , et sa négation ("<>") rencontrent par contre un vif succès: les élèves les ont en effet utilisé dans la construction de conditions sur des objets numériques ou chaînes.
- les enregistrements des entretiens montrent que la signification des expressions à valeur logique n'est pas claire pour les élèves. Une expression logique est comprise comme une "condition", c'est-à-dire une forme linguistique n'ayant pas de valeur en soi, mais impliquant selon les valeurs de ses composants, un comportement du dispositif ; dans cette conception, il est clair qu'une expression booléenne ne peut avoir pour eux de sens en dehors d'une expression conditionnelle, par exemple une alternative ; l'affectation d'une expression logique à une variable booléenne, ainsi que l'emploi de connecteurs propositionnels entre variables booléennes se heurte à cette conception.

3.4.3 L'apport de l'activité de résolution; les difficultés dues au manque de méthode de traitement des expressions logiques

- le programme à compléter que nous avons proposé aux élèves pour écrire leur version booléenne a été une incitation puissante à employer les booléens, en résolvant pour eux la question de l'entrée des valeurs logiques, et en leur donnant un schéma de résolution utilisant l'affectation sans alternative.
- dans l'ensemble les élèves semblent à même de franchir l'étape qui leur permet d'accéder au calcul sur les booléens : quatre programmes sont rendus comportant chacun au moins deux clauses programmées correctement. On se souviendra qu'il s'agit en majorité d'élèves de Première Scientifique, ayant surmonté en seconde les difficultés de la première alphabétisation.
- cependant, la programmation de la clause3, la plus complexe met en évidence l'absence de méthode de résolution qui permettraient aux élèves plus de sécurité dans leurs constructions: les transformations formelles, même les plus simples, ne sont pas employées ; ainsi, l'expression `not(Marie différent de Marc)` n'est pas transformée, malgré une simplification évidente. Seuls deux élèves donnent une expression correcte pour la clause3.
- face à cette absence de méthode, les enregistrements montrent deux comportements distincts. Jean-Manuel emploie l'énumération disjonctive des "cas possibles" ; ce comportement laisse persister deux erreurs dans le calcul de Clause3. Jérôme se tient plus prés de l'énoncé, en énonçant les cas d'impossibilité. On aperçoit les méthodes qui permettraient à ces élèves de systématiser leurs démarches: tables de vérité dans le cas de Jean-Manuel, transformations formelles à l'aide des théorèmes logiques dans le cas de Jérôme.

4 La résolution du second problème (VILLAGE)

(Texte complet en annexe 11 § 2: *programmer le parcours dans un village dont le plan est schématisé dans l'énoncé : un parcours sous forme d'une suite de bâtiments étant entrée par l'utilisateur, l'ordinateur doit indiquer si le parcours est possible*).

4.1 La recherche sur papier

Elle occupe 20 minutes à la fin de la première séance ; les élèves sont confrontés à la difficulté suivante : ils savent "le faire à la main", mais "ils ne voient pas comment l'ordinateur pourrait le faire". Seuls trois protocoles portent la trace d'une recherche; ils sont décrits en annexe 11 (§7), et nous les analysons ci-dessous :

4.1.1. Protocole C4 (Hervé)

La codification du plan choisie par Hervé consiste en un tableau de nombres et non de booléens. Si l'on corrige l'ambiguïté sur la valeur 0, comme élément du tableau, cette codification pourrait convenir : il suffirait, pour corriger, d'affecter un numéro ne correspondant pas à un bâtiment (par exemple 5, ou -1) aux éléments du tableau qui ne correspondent pas à un chemin. Pour le problème posé, cette codification permet un accès moins direct à l'information que la codification comme tableau à double entrée indexé sur les bâtiments, et à valeur booléenne proposé en solution ci-dessus ; en effet, pour savoir si un chemin existe du bâtiment n°N au bâtiment n°M, il faut tester `TAB[N,1]` , `TAB[N,2]` et `TAB[N,3]`. Elle se justifierait davantage pour des problèmes où il faudrait pouvoir connaître au moindre coût les points qu'il est possible d'atteindre à partir d'un point donné : par exemple lorsque le graphe est un arbre, et qu'il est important de connaître les "fils" de chaque sommet. Elle constitue néanmoins un témoignage de la capacité d'invention de l'élève : il résout la question de la codification des données , même si sa solution ne permet pas un accès optimal.

4.1.2. Protocole C5 (Denis)

L'analyse, sous forme d'un texte en français est constituée d'une série de "vérifications" à faire opérer par le dispositif, et précise le type des objets, mais, la codification du schéma n'est pas envisagée par cet élève et, en conséquence, l'analyse de cet élève ne dit rien sur la façon dont les "vérifications" peuvent être faites.

4.1.3. Protocole C7 :(Alain)

Ce protocole témoigne d'une tentative infructueuse d'écriture d'un programme à l'aide d'alternatives en chaîne.

4.2 Le travail sur ordinateur

Les élèves ont assez peu travaillé sur ce problème lors des deux séances sur machine. Hervé a écrit un programme utilisant la codification qu'il avait imaginé lors du travail sur papier (protocole C4) que nous analysons ci-dessous (protocole D4 version spontanée). 3 élèves, dont Hervé, ont ensuite travaillé à compléter un programme que nous leur avons proposé, et qui utilise une codification du plan du village à l'aide du tableau de booléens.

4.2.1 Un programme utilisant une codification du plan du village à l'aide d'un tableau d'entiers

Ce programme (protocole D4 version spontanée, décrit en annexe 11 §8) est écrit par Hervé, en conformité avec l'analyse produite lors de la séance de travail sur papier (protocole C4). La seule erreur du programme est l'oubli du 5ème trajet élémentaire (qui doit ramener à la maison, numérotée 0). Du fait de cet oubli, l'ambiguïté signalée dans l'analyse (0 dans le tableau désigne à la fois un trajet conduisant à la maison, et l'absence de trajets) est sans conséquence. On remarquera bien-sûr l'habileté de cet élève pour l'emploi des alternatives, mais aussi que la codification des données qu'il construit, ainsi que le calcul de la possibilité du trajet lui permettent *d'éviter totalement l'emploi des booléens.*

4.2.2 trois programmes écrits à la suite d'une incitation à employer une codification du plan du village à l'aide du tableau de booléens

Lors de la seconde séance sur ordinateur, nous avons proposé aux élèves, comme pour le premier problème (INVITATION), un programme à compléter :

```
program village;
 {complétez ce programme pour résoudre le problème 2}
 {le programme est répertorié VILLA.PAS sur la disquette}
 type maison=0..4;
     t_prom=array[0..3] of maison;
 var chemin:array[maison,maison]of boolean ;
 procedure init;
 var i,j:maison;
 begin
```

```
    for i:=0 to 4 do
  for j:=0 to 4 do  chemin[i,j]:=false;
    chemin[1,0]:=true;
    chemin[0,1]:=true;
    chemin[3,0]:=true;
    chemin[0,3]:=true;
    chemin[0,4]:=true;
    chemin[4,0]:=true;
    chemin[1,2]:=true;
    chemin[2,3]:=true;
    chemin[1,3]:=true;
    chemin[3,1]:=true;
    chemin[3,2]:=true;
    chemin[2,4]:=true;
    chemin[4,2]:=true;
end;
```

La question de la codification des données est ici résolue: le tableau à double-entrée `chemin` est à valeur booléenne ; `chemin[M,N]` prend la valeur vraie si dans le plan, il existe un chemin du bâtiment n°M au bâtiment n°N. La tâche des élèves est donc de programmer *l'entrée* d'un trajet, et le *calcul* de la possibilité de ce trajet. Ayant, par exemple, entré le parcours sous forme de 4 variables numériques `etape1, etape2, etape3, etape4`, il est aisé d'affecter une variable booléenne `possible` par la ligne suivante :

```
possible:= chemin[0,etape1] and chemin[etape1,etape2] and
           chemin[etape2,etape3] and chemin[etape3,etape4] and
           chemin[etape4,0]
```

4.2.3. Les programmes produits par les élèves.

Nous donnons en annexe 11 §8, l'écriture laissée par chacun des trois élèves ayant complété le programme. Nous analysons ici chacun de ces protocoles :

Protocole D1

Ce programme est très proche de la solution que nous avons donnée lors de la présentation de l'énoncé (variante non itérative). La procédure `EntréeDesDonnées` réalise non seulement l'entrée des données, mais aussi le calcul de la valeur de vérité du résultat. Ce calcul utilise pleinement la structure booléenne ; le résultat est affecté à une variable globale d'identificateur `"VRAI"` (ainsi, la variable `vrai` peut prendre la valeur `false` !) ; seul l'affichage du résultat met en jeu une alternative.

Protocole D4 version booléenne (Hervé)

Le caractère booléen des éléments du tableau chemin est reconnu et utilisé par cet élève. Par contre, l'élève utilise une chaîne de caractères pour le résultat, cette chaîne de caractères pouvant prendre 5 valeurs, une affirmative, et quatre négatives. On retrouve ici la forme "remplacement conditionnel" que cet élève a employé dans sa version précédente.On peut donc considérer que, comme dans plusieurs protocoles de résolution du premier problème (VILLAGE), l'élève recrée ici, à l'aide d'alternatives, une structure proche des booléens utilisant un type connu antérieurement. Par ailleurs, le dernier segment `chemin[d,0]` n'est pas vérifié (erreur déjà rencontrée dans la version spontanée).

Protocole D7 (Alain)

Comme en témoigne le choix de l'identificateur de la variable résultat, ce protocole résulte d'un travail en commun avec l'auteur du protocole D1. L'élève tente probablement d'explorer itérativement les segments d'un chemin pour repérer ceux qui ne sont pas possibles ; en fait, la recherche itérative porte sur TOUS les segments imaginables du village, et non sur les segments d'un chemin entré par l'utilisateur, la variable résultat prenant la valeur du dernier segment exploré. D'ailleurs, malgré l'identificateur de la procédure (`question`), il n'y a pas d'entrée d'un parcours par l'utilisateur. Il est donc possible que l'élève ne distingue pas bien le tableau `chemin`, comme codification du graphe de l'énoncé, et la donnée d'un PARCOURS à tester ;

l'identificateur "chemin", que nous avons choisi pour le tableau représentant le graphe, peut d'ailleurs contribuer à cette confusion. Pour cet élève, il n'y donc pas de compréhension de la codification du graphe proposée dans le programme donné à compléter.

4.3 Conclusions sur le second problème (VILLAGE)

4.3.1 Les élèves ont généralement reconnu la question de la codification du plan du village comme un problème mais ne l'ont pas résolu

- un élève a tenté une solution sans cette codification et a dû abandonner,
- un élève a produit une codification et un programme exécutable, tout en évitant l'emploi du type booléen,
- les autres n'ont produit ni codification ni programme.

4.3.2 Après que nous leur ayons fourni un programme à compléter, comportant une codification du plan à l'aide d'un tableau de booléens, seuls trois élèves ont écrit un programme dont deux manifestent une compréhension de cette codification

- le programme d'un élève montre une incompréhension manifeste de la codification proposée: il y a en effet confusion entre le tableau représentant le plan du village, et le trajet dont il s'agit d'évaluer la possibilité,
- un autre élève utilise la codification en l'ayant comprise, mais n'utilise pas le type booléen pour le résultat: son programme utilise la forme "remplacement conditionnel" qui évite le recours au OU logique.
- le troisième fournit un programme proche de la solution que nous avons exposée.

4.3.3 L'interprétation des difficultés des élèves

La codification d'un graphe par un tableau de booléens "ne va pas de soi". Proposée en solution, elle est comprise par certains et pas par d'autres.

Les conduites des élèves sont ils liés au seul manque de familiarité avec le type booléen, mis en évidence avec le problème INVITATION? Ce manque de familiarité s'est surtout traduit par des dificultés avec les connecteurs logiques (autres que ET) et des difficultés d'interprétation de l'affectation. Par contre, certains élèves ont employé de façon spontanée dans ce problème un vecteur (tableau à une entrée) de booléens pour représenter une "invitation"; on pourrait donc penser que, malgré leur manque de familiarité avec ce type de données, la conception et l'utilisation d'un tableau de booléens est à leur portée. La difficulté dominante tiendrait donc à la modélisation du problème. Nous avions fait l'hypothèse d'une difficulté spécifique aux problèmes de l'axe 2 auquel appartient VILLAGE: l'axe 2 est celui où les booléens servent à codifier non des **objets du problème**, mais des **relations entre ces objets**. Nous vérifions ici que, dans la seule codification obtenue de façon spontanée pour le second problème (VILLAGE), celle du protocole D4, un tableau est employé, mais ses éléments sont des *objets du problème* (les batiments du village, représentés par des nombres). Aucune utilisation des booléens, qui, dans ce problème, codifieraient des relations, n'est proposée de façon spontanée. Ces deux observations conduisent à considérer comme réaliste l'hypothèse d'une difficulté inhérente au type de modélisation impliqué par les problèmes de l'axe 2; le petit nombre de protocoles recueillis et l'absence d'entretien enregistré ne permet cependant pas de mieux la cerner.

Conclusions de la partie IV

comparaison avec les résultats obtenus pour les chaînes de caractères

1. Le choix du type des données

Nous avions noté chez les élèves confrontés à l'écriture de programmes utilisant les fonctions sur les chaînes une difficulté que nous avions noté *D.Type* (chapitre 6): ces élèves choisissaient spontanément un type selon la forme externe des données, et non selon le jeu de fonctions qu'ils souhaitent leur appliquer.

Nous notons ici dans le problème INVITATION que de façon spontanée, les élèves choisissent d'abord le type booléen puis l'abandonnent pour adopter une codification par un type numérique ou des chaînes, et une forme de programmation utilisant des alternatives. Ici aussi, c'est l'aspect externe des données qui guide le choix initial des élèves en faveur du type booléen; ce choix, judicieux ici, est écarté ensuite à cause du manque de familiarité des élèves avec le type booléen. Dans le problème VILLAGE, le choix d'un type pour représenter le plan du village est également un point important de la recherche d'une solution. Il est compliqué par le fait qu'il s'agit d'un type structuré. La seule solution produite de façon spontanée utilise un type structuré, mais élimine le type booléen comme type de base.

Dans le problème CHIFFRE (chapitre 6), les élèves pensaient spontanément les données comme numériques, mais réalisent le traitement comme un calcul de chaînes. Ici, dans le problème INVITATION, les élèves pensent spontanément les données comme booléennes, mais ne peuvent construire un calcul sur ces booléens. Les problèmes que nous avons présentés, aussi bien dans le cas des booléens que dans celui des chaînes de caractères, mettent donc en jeu la coordination entre la représentation mentale des données (qui se traduit par le choix spontané du type) et l'idée générale (le plan) du traitement que le sujet prévoie: il s'agit de passer de la question *Quelle est la nature des données* à la question *Que va t'on faire des données*. Cette coordination constitue pour nous une première manière de poser la question de l'abstraction des données. En effet, le plan, s'il est conçu à partir d'un traitement familier (dans les objets du monde) n'impose pas de typage des objets, et **l'élève, s'il comprend le typage comme une nécessité syntaxique sans rapport avec le traitement, va en fait typer les données selon leur forme externe**. C'est seulement au moment du raffinement du plan imposé par la programmation, que la question de la coordination va se poser. Cette coordination s'effectue d'une manière qui peut dépendre du problème: dans CHIFFRE, dans INSEE, le traitement sur les chaînes semble le plus adapté, et impose donc le type des données. Dans INVITATION, le choix spontané du type est le plus judicieux, la construction d'un traitement adapté impose aux élèves une amélioration de leurs représentations liées à ce type. VILLAGE impose une coordination objets-traitements remettant en cause de façon plus radicale les représentations spontanées: en effet, la construction d'un traitement adapté au dispositif conduit à considérer comme des données, non plus les objets intuitifs du problèmes, mais les relations entre objets.

Nous avons montré que, dans le cas des chaînes de caractères, la possibilité pour les élèves d'effectuer cette coordination objets-traitements est liée à leur capacité à envisager qu'un même objet (un nombre dans les problèmes que nous avons proposé) puisse avoir des codifications distinctes pour le dispositif, et à la disponibilité de fonctions de conversion de type. Dans le cas des booléens étudié dans ce chapitre, les capacités de l'élève concernant le calcul sur les booléens semblent en jeu pour les problèmes concernant les objets à deux états (par exemple INVITATION), et la capacité de passer d'une modélisation des objets à la modélisation de leurs relations semble déterminante dans le problème VILLAGE qui se ramene à un parcours de graphe.

2. L'intégration différenciée des connecteurs logiques

Nous avions répertorié sous le nom *D.Sem.Cha.* les difficultés liées à la non-constitution d'un S.R.T. propre chaînes de caractères, et observé la façon dont elles évoluent au cours des apprentissages. L'observation dans le cas des booléens étant menée de façon moins approfondie, nous ne tenterons pas une typologie complète de ces difficultés, et nous n'avons pas les moyens de noter des évolutions. Nous avons observé une difficulté liée à l'emploi des connecteurs propositionnels et aux transformations d'expressions booléennes:

- les connecteurs = et <> (différent) sont utilisés sur des booléens par la majorité des élèves dans le problème INVITATION.
- le connecteur conjonctif AND est utilisé par la majorité des élèves; cependant un élève construit des expressions à l'aide d'alternatives en chaîne qui lui permettent d'éviter ce connecteur.
- l'utilisation d'une alternative complète permet généralement aux élèves d'éviter le recours à la négation.
- le connecteur disjonctif OR n'est pas utilisé, sauf dans le cas particulier de la forme disjonctive canonique.

En tant que fonctions sur les booléens, les connecteurs propositionnels sont donc conçus de façon plus ou moins assurée: l'égalité et la différence s'appuient sans doute sur une bonne intégration de l'équivalence logique (l'élève, même s'il a des difficultés à donner un statut précis aux quantités booléennes, peut considérer que l'égalité de deux variables booléennes traduit le fait qu'elles "représentent la même chose", donc sont vraies en même temps); de plus, ces connecteurs ont été utilisés pour les types numériques et chaîne de caractères. De même, le AND est relativement intuitif, et il a probablement été abondamment utilisé dans la partie "condition" des instructions conditionnelles. La négation a au contraire été peu utilisée dans les instructions alternatives; en effet, il est généralement possible d'éviter la négation dans ces instructions, soit en utilisant le connecteur "différent de", soit en échangeant les branches de l'alternative[143]. Les élèves sont manifestement gênés par les négations d'expressions contenant des connecteurs disjonctif ou conjonctif. Ils sont gênés également par l'ambiguïté sur le sens du connecteur OR (exclusif ou non exclusif) et par les propriétés de ce connecteur relativement aux autres; dans la forme disjonctive canonique, ces questions ne se posent pas, ce qui explique sans doute que c'est le seul cas d'emploi du connecteur OR. Dautre part, la programmation d'instructions alternatives n'impose pas l'emploi de ce connecteur: plusieurs élèves disposent d'une forme de programmation que nous avons qualifiée de "remplacement conditionnel" qui leur permet de programmer efficacement sans le connecteur disjonctif[144].

3. La compréhension des expressions booléennes comme "conditions"

La programmation par calcul sur des booléens ne va pas de soi: les élèves doivent passer d'une conception où des expressions logiques au statu flou forment la partie "condition" des instructions conditionnelles, à une conception où les booléens sont des entités calculables au même titre que les nombres et les chaînes. Le fait que les "entrées-sorties" ne sont pas réllement possibles sur ce type, joue un rôle dans cette

---

[143] Par contre, la maîtrise de la négation peut-être critique pour l'itération; en particulier, l'usage de la forme TANT QUE... impose de former une condition de continuation par négation de la condition de sortie.

[144] Dans les tout premiers apprentissages, on peut rencontrer des difficultés avec des écritures alternatives telles que IF X=1 OR X=2, les sujets restants fixés sur l'écriture IF X=1 OR 2, ce qui indique qu'en fait, pour eux, la partie condition code une forme verbale naturelle et non une expression booléenne. On remarrquera que le connecteur AND n'est pas concerné par ce type de confusion des niveaux d'écriture.

préhension du type booléen comme "un type à part", et il se révéle nécessaire d'appliquer aux élèves des procédures permettant de compenser cette difficulté pour qu'ils puissent envisager un programme comme calcul sur des booléens. En programmation impérative ceci implique que des expressions booléennes peuvent être l'objet d'une affectation. Nous avons pu observer que l'affectation du résultat d'une expression booléenne à une variable de ce type nécessite une interprétation particulière, distincte de l'affectation portant sur une variable d'un autre type. Initialement, pour les élèves, une expression booléenne représente une **"condition" qui a un effet en tant que partie d'une expression conditionnelle, et non un calcul dont le résultat peut prendre les deux valeurs booléennes**, d'où la difficulté à l'utiliser dans une affectation. Une autre manifestation de cette difficulté, observée chez des étudiants plus avancés est la persistance de l'affectation des variables booléennes par *alternatives*: `if <condition> then booleen:= true else booleen:= false` au lieu de l'affectation *directe*: `booleen:=<condition>`. Il serait intéressant d'approfondir l'étude de ce point, en particulier auprès d'élèves présentant des difficultés d'acquisition.

4. Comment les élèves peuvent-ils progresser dans l'emploi de booléens?

Contrairement aux booléens, le traitement des chaînes de caractères, que nous avons étudié dans les chapitres précédents,ne relève pas d'un champ scolaire extérieur à l'informatique. Certes, la structure numérique, faisant l'objet d'apprentissage en mathématiques est présente, mais le double aspect (cardinal, ordinal) des nombres entiers, qui interagit fortement avec les représentations des élèves, ne fait pas l'objet, au plan scolaire, d'un traitement particulier, une fois que cette structure numérique est considérée comme acquise (c'est-à-dire après la première année de l'enseignement obligatoire). On ne peut donc attendre des acquisitions en dehors d'un travail en informatique. Par contre, les propriétés des booléens, les transformations sur les expressions booléennes constituent un champ de connaissance scientifique parfaitement identifié: il s'agit de la logique propositionnelle du premier ordre. Nous distinguons deux types d'acquisition dans ce domaine qui permettraient aux élèves de progresser dans l'emploi des booléens:

- des capacités logiques mises en oeuvre par exemple dans un raisonnement en Mathématiques.
- les connaissances explicites en Algèbre de BOOLE, pouvant servir de support à des méthodes d'écriture des fonctions logiques (régles de transformations d'expressions, tables de vérité...).

On peut donc s'attendre à ce que des acquisitions de l'un ou l'autre type facilitent les acquisitions concernant l'emploi des booléens en informatique. Qu'en est-il des élèves que nous avons observé? Il s'agit d'élèves de classes scientifiques, ayant passé avec succès le cap de la première alphabétisation en informatique. Il n'est pas étonnant qu'ils disposent de facilités du point de vue du raisonnement, ce qui leur permet de considérer des expressions logiques. Mais, raisonner n'est pas calculer des expressions logiques, et nous avons pu vérifier qu'ils ne connaissent ni les tautologies classiques ni les tables de vérité, et qu'ils ne font pas de transformation des expressions, même simples. Il serait intéressant de vérifier si des méthodes de constitution ou de transformation des expressions logiques, fondées sur des connaissances en Algèbre de BOOLE, peuvent les aider à progresser vers une meilleur représentation des expressions logiques, une meilleure intégration des connecteurs et donc vers la capacité à envisager d'emblée un traitement sur des quantités booléennes.

Conclusion générale

Chacune des quatre parties de cette thèse se termine par un chapitre de conclusions permettant de marquer l'évolution des idées au cours de la recherche. A partir de ces conclusions, nous reprenons ci-dessous thème par thème, les résultats obtenus et les questions ouvertes au long des différents chapitres; nous partons des **systèmes de représentation et de traitement** qui, au long de la thèse, nous ont servi d'outil pour observer et caractériser les conduites des élèves en situation de résolution; puis nous faisons un bilan des apports de la réflexion menée sur les **énoncés de problèmes**. Nous avons posé au début de cette thèse des questions sur un type de **situation pédagogique**: nous indiquons les éclairages obtenus, et les questions nouvelles ouvertes. **L'intervention des structures de données dans les apprentissages en informatique** est le thème dans lequel s'inscrit cette thèse: nous faisons le bilan des aspects de cette question abordés dans cette recherche, et des perspectives ouvertes. Enfin, nous montrons quelles directions de travail il est possible d'établir à partir de cette thèse, pour la définition de cursus s'adressant à des débutants.

1. Les systèmes de représentation et de traitement

Un S.R.T. est pour nous une "machine mentale" qui permet à un sujet donné, à un moment précis d'une tâche de construire une solution ou de vérifier sa validité. Il s'observe à partir des conduites du sujet en situation de résolution. Un sujet peut disposer de différents S.R.T., pour une même tâche ou pour des tâches proches, plus ou moins différenciés et organisés. En résolvant, il crée des codes de passages d'un S.R.T. à l'autre.

1.1 La construction de représentations mentales par le débutant

Selon notre première hypothèse, beaucoup de débutants ne situent pas leur activité de résolution d'emblée au niveau des concepts. Chez nombre d'entre eux, la construction de représentations mentales adéquates du dispositif, s'organisant en S.R.T. différenciés et mis en relation, est un processus à la fois lent et nécessaire pour l'émergence des concepts. Ce processus implique un travail de mise en évidence et de remise en cause de S.R.T. non adaptés aux traitements informatiques.

L'ampleur des difficultés rencontrées par les débutants dans l'emploi des fonctions qui définissent les chaînes, ainsi que pour l'utilisation sur des chaînes des éléments de base du langage (affectation, notation fonctionnelle) se confirme: pour une bonne moitié des élèves observés, les écritures dans les programmes servent en fait à **"donner à l'ordinateur" toutes les informations sur le résultat souhaité, sans que ces informations soient conçues comme des entités calculables, et sans que que le traitement soit conçu comme un calcul.** Nous trouvons bien chez ces élèves des **conceptions *naïves*** où les objets et le traitement par le dispositif ne sont pas différenciés de l'environnement habituel. Les difficultés sont plus fréquentes dans les calculs qui sollicitent la structure de chaîne dans sa généralité (par exemple une sous-chaîne qui ne soit ni le début ni la fin d'une chaîne). Les élèves qui manifestent un minimum de compréhension de l'utilisation de la structure de chaîne sont quasiment tous orientés en Première Scientifique. Mais la réciproque n'est pas vraie, en ce sens que près de la moitié des élèves orientés en Première Scientifique échoue à une épreuve comportant des problèmes simples impliquant l'emploi des chaînes de caractères.

Nous avons pu constater que les représentations des éléments de base du langage comme **l'affectation et la notation fonctionnelle sont étroitement liées, dans les S.R.T., aux représentations des objets**. Ainsi, dans les chaînes, le rôle de l'affectation peut être ignoré, les écritures fonctionnelles et/ou les instructions de sortie étant comprises comme agissant sur les objets internes. De

même les élèves conçoivent les "conditions" (c'est-à-dire les écritures fonctionnelles à valeur booléenne) comme des formes verbales sans statut d'objet informatique et donc ne donnent pas de signification aux calculs, aux entrées-sorties et aux affectations portant sur ces écritures.

Certaines représentations concernent en lui-même le type de donnée que les élèves utilisent. Par exemple, dans les **représentations naïves des chaînes**, les deux derniers arguments de la fonction sous-chaîne sont compris comme un ensemble vague d'indications incluant des éléments d'ordre sémantique, alors que dans une conception correcte de cette fonction, ces arguments ont chacun un statut précis qui permet au dispositif de *calculer* la sous-chaîne. Nous observons également que, dans l'utilisation des chaînes, le **rapport entre identificateur de variable et objet** peut ne pas être clair: par exemple, l'affectation à une variable `A$` d'un élément calculé d'une chaîne `C$` ne conduirait pas à la duplication de la valeur de cet élément, mais à ce que la variable `A$` représente effectivement cet élément, qui ne cesse pas pour autant d'être une partie de `C$`; c'est ce que nous avons appelé la ***représentation des objets comme entités physiques non duplicables***.

Mais, cette conception correcte est elle-même *compatible* avec la représentation naïve: en effet le couple des arguments dans cette conception donne bien "toutes les informations" permettant de définir une sous-chaîne.C'est pourquoi, à travers certains exemples ou exercices, l'élève peut ne pas avoir conscience du statut précis de ces informations. Il en résulte qu'une activité autonome du sujet, dans des problèmes spécifiquement construits est nécessaire pour remettre en cause la représentation.Ici aussi, dans beaucoup de problèmes, une résolution correcte peut se révéler *compatible* avec cette représentation erronée, et donc cette représentation ne peut se traduire par des conduites observables que dans des problèmes spécifiques: ceux où, par exemple, une concaténation porte sur des parties calculées d'une même chaîne (par exemple les problèmes ONJOURB et RONJOUB): dans ce cas, la concaténation est comprise comme un déplacement à l'intérieur de la chaîne (et non un "recopiage"), ce qui produit des effets observables (BONJOURB au lieu de ONJOURB dans le problème du même nom).

Dans **l'utilisation des booléens**, les élèves se révèlent peu à l'aise avec certains connecteurs propositionnels (négation, disjonction), alors que pour d'autres (égalité, différence, conjonction), ils peuvent transférer des représentations construites antérieurement pour l'écriture de la partie "condition" des alternatives, elles-mêmes issues de représentations naïves de l'alternative. Ici la difficulté est une **représentation mentale trop restreinte du type booléen**: l'introduction de nouveaux connecteurs ne pourrait se faire que par une clarification de leurs rapports avec ceux pour lesquels une représentation est déjà installée. La question de cet élargissement des représentations des booléens reste ouverte.

Les difficultés observées avec **l'utilisation des ordinaux** s'apparentent à celles rencontrées avec les booléens, en ce sens qu'il s'agit d'entités que les élèves n'ont jusqu'ici rencontrées ni dans des calculs, ni surtout dans des expressions algébriques. Mais le contexte du traitement de chaîne intervient dans les représentations mentales des ordinaux. Ceux-ci sont en fait peu distingués des caractères qu'ils pointent dans une chaîne: par exemple, la variable `I` qui a reçu comme valeur le rang d'un caractère d'une chaîne, peut, dans la compréhension de nombreux élèves, coder aussi le caractère lui même; `L$` ayant pour valeur un caractère d'une chaîne, `L$+1` codera le caractère suivant dans la chaîne[145]. Des représentation erronées des quantités numériques intervenant dans la structure de chaîne peuvent également être mises en relation avec la représentation naïve du langage: par exemple, dans une chaîne de longueur `L`, l'ordinal `L-1` s'observe souvent comme rang du dernier caractère, et le cardinal `L-1` s'observe comme longueur d'une chaîne (commençant pourtant au rang 2) ne comprenant pas ce dernier caractère; dans les deux cas `L-1` code l'action: ôter le dernier caractère; ce sont bien les écritures

[145] Plus généralement, il s'agit d'une difficulté à distinguer adresse et contenu qui pourrait se traduire également dans l'utilisation des tableaux par la confusion indice élément.

algébriques qui se voient attribuer un rôle dans le changement d'état du dispositif. Ce type d'erreur est significatif d'une **conjonction de représentations du langage, des chaînes et des entités numériques** qui interviennent dans cette structure. Cette conjonction confirme que les représentations des éléments généraux du langage et des éléments particuliers à une structure sont totalement imbriquées dans les S.R.T.; il n'est donc pas étonnant que les débutants ne puissent pas spontanément mettre en relation les éléments généraux, d'un domaine de tâche à l'autre, et donc raisonner au niveau des concepts.

1.2 Dans certains problèmes, les réponses exactes peuvent être *compatibles* avec des représentations erronées

Nous avons rencontré le phénomène suivant dans toutes les observations: l'élève répond de façon exacte à un problème mettant en jeu sa compréhension d'une structure de donnée, puis dans un autre problème, sa réponse témoigne au contraire de représentations totalement inadaptées. Il se trouve en fait que, dans le premier problème, la réponse exacte peut s'obtenir avec la représentation inadaptée. Par exemple, le troisième argument de la fonction sous-chaîne, si l'on souhaite calculer la sous-chaîne d'une chaîne de longueur `L`, privée de son dernier caractère est bien `L-1`. Dans une conception correcte, cet argument est la longueur de la sous-chaîne souhaitée, mais, pour l'élève, elle peut fort bien coder l'action d'ôter le dernier caractère, dans une représentation où, comme nous l'avons vu ci-dessus, les écritures fonctionnelles codent des actions. Il en résulte que **les premiers exemples auxquels l'élève est confronté peuvent en fait conforter des représentations spontanées** qui deviennent ainsi particulièrement résistantes. Pour que ces représentations produisent des effets susceptibles de conduire l'élève à adapter ses représentations, il est nécessaire que celui-ci soit confronté à des problèmes où la structure de données est sollicitée dans sa dimension la plus générale (dans l'exemple ci-dessus, pour le calcul d'une sous-chaîne ne commençant pas au premier caractère); le plus souvent, ces problèmes doivent être construits par l'enseignant ou le chercheur en fonction d'une hypothèse sur les représentations spontanées.

1.3 Représentations des objets et représentations des traitements

Nous avons voulu que les S.R.T. puissent servir à décrire, dans la résolution d'un problème donné par un sujet donné, **l'interaction des représentations du sujet relatives aux objets avec ses représentations des traitements**, de façon à prendre en compte par exemple, le fait que le sujet éprouve généralement des difficultés à transférer d'un domaine de tâche à l'autre des capacités algorithmiques. C'est pourquoi nous avons postulé l'existence et l'interaction, dans un S.R.T. donné, d'une part de *plan(s) relatif(s) aux objets*, et d'autre part, de *plan(s) relatif(s) aux traitements*. Les premiers comprennent, en plus des représentations des entités qui codifient les objets du problème, les représentations des éléments de base du langage (affectation, notation fonctionnelle...), qui, nous l'avons vu ci-dessus, leur sont étroitement liées. Les plans relatifs aux traitements interviennent lorsque le sujet construit un traitement élaboré (alternatives imbriquées, itération...).

Dans les deux chapitres où nous explorons l'interaction de ces deux types de plans dans des S.R.T. relatifs à l'itération sur une chaîne de caractères, les réponses des élèves s'analysent comme résultant d'une **conjonction de représentations relatives aux objets, et de représentations relatives aux traitements** souvent influencées par une conception naïve du traitement de données textuelles. Les représentations qui ne différencient pas les ordinaux des caractères s'observent en effet en conjonction avec une représentation du dispositif comme capable d'accéder de façon directe (ou associative) à la position d'une sous-chaîne donnée dans une chaîne. Il en est de même de la représentation "incantatoire"[146] de l'affectation, qui conduit

[146] Dans la représentation "incantatoire" de l'affectation, cette instruction code un état que

des élèves utilisant la forme d'itération REPETER...JUSQU'A...à ne pas distinguer condition de sortie d'itération et affectation. Egalement, la conception naïve du langage s'observe en conjonction avec la non-intégration du rôle des éléments de l'itération (avertisseurs d'itération, compteur...): ainsi, les avertisseurs d'itération peuvent être présents, sans qu'il y ait compréhension du mécanisme d'exécution, en particulier l'évolution du compteur; en effet la représentation des écritures fonctionnelles sur les chaînes comme codage d'actions permet que ce mécanisme soit ignoré. Nous avons noté là aussi, que ces conjonctions de représentations peuvent fort bien ne pas être remise en cause par certains problèmes, et que, de façon générale, elles conduisent à des points d'équilibre particulièrement résistants.

1.4 Evolution des représentations

Nous résumons ci-dessous quelques points sur lesquels, au cours d'une observation longitudinale, nous avons pu constater une évolution des représentations.

1.4.1 Le typage des données.

Dans les langages utilisés, le type des objets est précisé par une déclaration explicite; à chaque type correspond un jeu de fonctions spécifiques, et des fonctions de conversion de type permettent pour une même entité, le passage du codage dans un type à un codage dans un autre type. Les élèves peuvent donc avoir à faire le choix d'un type de données pour le codage d'objets du problème, et il faut que ce codage soit cohérent avec le jeu de fonctions que, dans une analyse souvent implicite, ils prévoient d'utiliser sur les objets codifiés. Nous avons montré que ce choix n'a rien de spontané: en effet les élèves choisissent le type en considérant la "nature" des données, plus que le jeu de fonctions qu'ils prévoient d'utiliser. D'où l'importance de problèmes remettant en cause ce choix spontané. Nous avons observé que, à l'issue d'un travail à partir de problèmes de ce type, des élèves, pourtant très en difficulté, maîtrisent le codage des nombres à la fois sous forme numérique et sous la forme de la chaîne de sa représentation en base 10 (cette maîtrise est mise en évidence dans le problème SOMME où il s'agit d'obtenir la somme des chiffres des centaines et des unités dans un nombre à trois chiffres), et noté que ce progrès a une influence positive sur leur représentations des objets informatiques. Cette observation contredit des hypothèses suivant lesquelles il conviendrait d'éviter au débutant les difficultés liées au type des objets; nous avons vu que cette hypothèse intervient de façon implicite dans la spécification du langage LOGO: en effet, dans ce langage, il n'y a pas de typage des objets, les fonctions sur les listes s'appliquant à tous les objets, au prix de certaines incohérences.

1.4.2 L'utilisation de l'affectation

A la suite de l'observation des difficultés des élèves en partie II, nous avions pensé que **l'utilisation de l'affectation pour la décomposition des calculs permet à l'élève, à la fois de mieux maîtriser l'affectation et d'éviter des écritures fonctionnelles trop complexes** auxquelles il lui est difficile de donner une signification, autre que naïve. Il nous semble qu'un élève capable d'utiliser ainsi l'affectation raisonne dans un S.R.T. déjà dissocié du S.R.T. directement déduit de l'expérience naïve, puisque ce S.R.T. inclue une certaine compréhension du calcul sur des objets. Dans l'observation longitudinale, nous avons donc orienté les élèves observés vers ce type de décomposition, et constaté que, à l'issue de cette observation, les élèves, à la différence de ceux sur lesquels portait l'observation ponctuelle, maîtrisent les écritures fonctionnelles, et donnent un sens à l'affectation. Notre choix pédagogique se révèle donc positif pour ces élèves, choisis parce qu'ils

---

l'élève souhaite voir réalisé, et non un mécanisme par lequel une variable voit sa valeur évoluer. Par exemple, l'affectation syntaxiquement correcte: `LET L$=MID$(MOT$,X,1)` peut pour l'élève désigner `X` comme le rang du caractère `L$` dans la chaîne `MOT$`, au lieu d'être comprise comme une instruction donnant une valeur au caractère `L$`

étaient initialement en grande difficulté.

Cette utilisation "intensive" de l'affectation met au jour d'autres éléments de représentation liés à l'affectation: c'est particulièrement le cas de la représentation "incantatoire" de l'affectation (voir note ci-dessus), qui se révèle particulièrement résistante à cause du phénomène de *compatibilité* que nous avons rappelé en 1.2.

1.4.3 Les représentations de l'itération

En fonction des résultats ci-dessus, nous avions pensé intéressant, dans le cadre du traitement itératif des chaînes, de **jouer sur la conjonction entre représentations des objets et représentations des traitements pour créer un déséquilibre**: nous avions constaté lors d'une observation ponctuelle, qu'une adaptation des représentations mentales des objets pouvait conduire les élèves à remettre en cause leur représentation du traitement directement issue de l'expérience sensible, et à construire un traitement adapté. Nous n'avons pas observé ceci de façon nette au cours de l'observation longitudinale. Nous avons attribué ceci à un défaut d'analyse préalable: l'intervention conjointe des représentations des objets et des représentations des traitements lors de la construction des traitements itératifs a présenté des caractères assez différents de ceux reperés dans l'observation ponctuelle, en grande partie à cause de la forme d'itération différente adoptée par cette classe (REPETER...JUSQU'A.... au lieu de POUR...FINPOUR). De ce fait, **les problèmes spécifiques ("recherche-translation") sont arrivés trop tôt dans la progression**, à un moment où nous n'avions pas encore caractérisé les représentations des objets pour ces élèves, et où ceux-ci n'avaient pas encore commencé à opérer des différenciations dans ce domaine.

1.5 Questions restant posées

1.5.1 Variété des représentations, dépendance par rapport au contexte

Les représentations mentales observées chez les élèves dans un contexte donné paraissent assez stables; elles évoluent, se différencient, mais leurs caractéristiques dominantes ne connaissent pas de transformation brutale. Par contre, les différences repérées dans les différentes observations, conduisent à penser que les représentations peuvent varier notablement en fonction du contexte. Nous avons mené cette thèse en considérant que les résultats obtenus seraient certainement dépendants du choix de la classe des langages impératifs, en tant que contexte, mais nous avons pris soin de ne pas introduire, dans les problèmes, des éléments qui seraient spécifiques à un langage particulier dans cette classe. Nous devons considérer maintenant, que cette précaution ne suffit pas à assurer un caractère suffisamment général aux représentations observées qui leur permettrait de fonctionner dans toutes les situations où les élèves utilisent un langage impératif. Il devrait donc exister une variété importante de représentations d'une même tâche, dans différents contextes, y compris dans la même classe de langage. Les observations de [DAGDILELIS BALACHEFF CAPPONI 91] vont plus loin dans ce sens, puisqu'elles montrent que les mêmes élèves, pour la même tâche, mettent en oeuvre des représentations non seulement différentes, mais aussi séparées de façon étanche, dans deux contextes de programmation (le langage PASCAL et le tableur MULTIPLAN) pourtant de la même classe impérative.

1.5.2 Elargir et classifier le corpus d'observations

La variété des représentations et de leur évolution chez les débutants ouvre donc un large champ de recherche: une première direction est l'élargissement du corpus d'observation et d'explicitation des représentations, de façon à opérer des classifications et des mises en relation; en particulier, les différentes formes de programmation: impératif, fonctionnel, logique, la programmation à l'aide de différents outils, tableurs, S.G.B.D., constituent autant de contextes où il serait important de disposer de ce corpus: par exemple, il serait intéressant de savoir comment se traduisent les difficultés concernant les objets dans un langage n'utilisant pas

l'affectation[147]. La connaissance des représentations et de leur évolution est en effet fondamentale pour orienter aussi bien la stratégie d'apprentissage dans son ensemble, que pour fonder l'intervention pédagogique ponctuelle auprès des élèves en situation de résolution.

1.5.3 Etudier le niveau des représentations

Une seconde direction est la recherche du niveau de hiérarchisation et de différenciation des représentations chez un sujet pour qu'il puisse opérer des transferts entre des représentations construites dans différents contextes: nous avons cherché à provoquer ces différenciations chez les élèves observés au chapitre 10, et nous pensons que leurs représentations ont évolué. Nous ne pouvons cependant dire dans quelle mesure le niveau atteint par ces élèves permet un transfert dans un autre contexte. De même l'observation de [DAGDILELIS BALACHEFF CAPPONI 91] porte sur peu de séances dans chacun des contextes, et on peut penser que les représentations mises en oeuvre sont en fait très proches de représentations spontanées.

2. Validité des problèmes choisis pour l'analyse des représentations et la construction d'une progression

Au chapitre 3, en fonction de notre analyse de l'activité de programmation dans les premiers apprentissages en informatique, nous avons proposé les spécifications suivantes, pour des énoncés de problèmes permettant à l'élève de faire progresser ses représentations en opérant des différenciations à partir de ses représentations naïves:

- le niveau où est posé le problème permet que les élèves considèrent de façon intuitive les objets en jeu et le lien entre données et résultats;
- le niveau du traitement par le dispositif informatique impose, quant à lui, une "codification" de ces objets sous forme de données informatiques et la production d'une solution comme "calcul" sur ces données codifiées;
- les problèmes font interagir des concepts informatiques de base et au contraire, évitent de trop faire intervenir des éléments conceptuels (par exemple mathématiques) en dehors du champ de l'informatique.

Les observations montrent que, en mettant à contribution les types de base (chaînes de caractères, booléens) des langages impératifs, il est possible de construire une variété de problèmes répondant, selon nous, à ces trois spécifications. Les problèmes ainsi construits se sont révélés des outils très utiles pour l'observation des conduites et l'analyse des représentations. Nous avons comparé par exemple, les résultats obtenus d'une part avec les problèmes de "recherche-translation" et d'autre part avec les problèmes plus classiques de "comptage d'occurrences": il se confirme que **les problèmes de "recherche-translation", construits en fonction des spécifications ci-dessus, font intervenir davantage la nécessité d'une structure algorithme compatible avec la structuration des données et conduisent donc davantage à la remise en cause de représentations spontanées**. Il a donc là aussi un important travail à poursuivre: analyse critique des énoncés existants, production de nouveaux énoncés...

Le retour critique sur l'analyse préalable à l'observation longitudinale, montre qu'**il reste beaucoup à faire dans la construction de progressions**: en particulier, il est important de se demander, pour chaque problème, si la réponse correcte n'est pas en fait *compatible* avec une représentation erronée, auquel cas la résolution n'apporterait pas de progrès significatif. Ce retour critique montre aussi que **l'organisation de ces problèmes en progression nécessite une analyse approfondie**; en particulier, une connaissance plus étendue des représentations

---

[147] Nous rejoignons donc [DUCHATEAU 91] quand il déclare «*Une liste des écueils rencontrés, des points délicats* (dans les approches fonctionnelles de la programmation) *ferait sans doute plus pour la reconnaissance de ces pratiques d'enseignement que les discours triomphalistes destinés à y faire adhérer de nouveaux partisans*»

spontanées permettrait d'éviter des erreurs de conception des progressions.

3. La situation pédagogique

Nous avons expliqué au chapitre 3 pourquoi les **situations de résolution par petits groupes, encadrées par l'enseignant, l'ordinateur étant à disposition pour des essais de compilation et/ou d'exécution** nous intéressent prioritairement. Par rapport à ce type de situations, nous avons trouvé pertinentes les questions suivantes:

- comment les contraintes d'expression et les effets en retour par lesquels se manifeste **la présence du dispositif informatique** (résultats d'exécution, messages d'erreurs à la compilation et à l'exécution) interviennent-ils dans la construction des représentations des élèves?
- selon quels axes peut-on classer les **interventions de l'enseignant?** Sur quels éléments l'enseignant peut-il s'appuyer pour interpréter les productions des élèves et orienter son intervention?
- comment se fait la **communication** dans le groupe, et avec l'enseignant ?

3.1 La prise en compte des représentations des élèves dans les interventions de l'enseignant et le rôle de l'erreur

Dès les premières observations (chapitre 6), nous constatons que l'intervention pédagogique auprès d'élèves en situation de résolution, si elle ne peut s'appuyer sur la connaissance précise de leurs représentations, porte tendanciellement sur les aspects syntaxiques, dans le but de conduire les élèves à un programme syntaxiquement correct puis d'obtenir une amélioration à partir des résultats d'exécution. **Nous avions noté qu'une intervention principalement syntaxique peut réussir auprès d'élèves disposant de représentations adéquates des éléments généraux du langage mais que d'autres élèves, dont les représentations sont très éloignées du fonctionnement du dispositif, n'obtiennent pas de meilleure compréhension en améliorant leur syntaxe, et ne peuvent interpréter les résultats d'exécution.**

Pour ces élèves, erreurs de syntaxe et résultats d'exécution erronés sont compris comme l'indication que "cela ne marche pas", sans qu'ils puissent intégrer dans leurs représentations les causes de ce non-fonctionnement. En effet, le niveau auquel se situent ces "feed back" du dispositif suppose dans la plupart des cas que l'élève soit proche d'une représentation adéquate. Dans cette situation, **c'est souvent à l'enseignant qu'il appartient d'interpréter cette réponse du dispositif.** Sans la connaissance de la signification pour l'élève de l'écriture erronée, il peut seulement rappeler les contraintes de la syntaxe, ou suggérer des améliorations sans rapport avec la conception de l'élève et qui restent par conséquent sans effet.

A la suite de l'observation plus longitudinale du chapitre 10, nous notons que, pour être efficace, **l'intervention pédagogique de l'enseignant auprès d'élèves présentant ces difficultés, nécessite une analyse des expressions produites par l'élève sur une période suffisamment longue, et sur des problèmes variés.** La représentation "incantatoire" de l'affectation (Cf note ci-dessus), par exemple, est dégagée seulement après plusieurs séances, trop tard en particulier pour qu'une intervention efficace soit possible sur les problèmes de "recherche-translation". Il semble donc que l'intervention enseignante ne puisse se passer d'une part de la connaissance du corpus concernant les représentations dont nous avons parlé ci-dessus, et d'autre part, d'une observation continue des productions de l'élève et de leur mise en relation avec ce corpus.

L'étude des représentations erronées tient une place importante dans cette thèse, et nous proposons précisément de fonder les interventions pédagogiques sur une connaissance approfondie de ces représentations. Cette direction de travail semble devoir être débattue entre chercheurs, comme en témoigne la réflexion suivante de [DUCHATEAU 91], faisant référence à une présentation de notre travail au

second colloque de didactique de l'informatique: «*Quel statut faut-il réserver à l'erreur: faute ou tremplin?... Je dois avouer que c'est l'une des questions essentielles que me posent des approches comme celles de LOGO, où l'erreur est érigée en principe et où la construction du savoir s'appuie sur les maladresses et les bévues*». Nous pensons avoir montré que les représentations des élèves ne peuvent être mises au niveau de la *maladresse* ou de la *bévue*. Réponses que l'élève déduit de son expérience sensible face à un problème de programmation, puis enrichit progressivement par la confrontation en situation pédagogique aux contraintes du dispositif, les représentations sont des *connaissances* de l'élève. En tant qu'enseignant d'informatique, nous pouvons témoigner que l'étude des représentations menée dans cette thèse a eu une influence directe sur notre façon d'enseigner en nous aidant à prendre en compte ces connaissances, au delà de la réaction spontanée de gêne que peuvent provoquer les erreurs dûes aux représentations erronées.

La prise en compte de l'erreur de l'élève, comme témoignage de représentations en train de se construire nous semble un point de passage obligé pour le type d'apprentissage étudié dans cette thèse, portant sur les objets manipulés dans le programme, et le lien entre les objets et les traitements. Il se pourrait que d'autres capacités en programmation, par exemple l'articulation de traitements complexes, soient moins liées à des représentations du sujet, et ne puissent pas, par conséquent se développer par un processus d'essai-erreur. Ce type de capacité supposerait donc au contraire l'enseignement de méthodes permettant de "programmer juste du premier coup". Mais nous avons souligné que **la mise en oeuvre par l'élève de méthodes peut être efficace seulement s'il dispose de systèmes de représentations et de traitements suffisamment élaborés** pour donner une signification aux éléments méthodologiques. La constitution de ces S.R.T. est donc pour nous à assurer en priorité dans un enseignement à des débutants, ce qui justifie l'importance que nous avons donnée aux représentations et procédures erronées.

3.2 Questions ouvertes

Nous avons tenté une analyse de la situation pédagogique à l'aide des éléments théoriques dûs à [BROUSSEAU 86]. Parmi d'autres apports intéressants de ces éléments théoriques, nous avons relevé la question de **la *dévolution* du problème aux élèves**. Si cette dévolution s'opérait correctement, l'élève devrait pouvoir produire des valeurs de test suffisamment générales pour appuyer son raisonnement de validité du programme. Or, dans bien des cas, l'élève produit des valeurs de test visant seulement à conforter son analyse, et c'est l'enseignant qui doit amener les valeurs qui montrent que l'analyse n'est pas générale. Cet exemple nous montre que l'analyse des situations pédagogiques en informatique, dans leurs différentes phases à l'aide d'outils théoriques, est une direction de recherche importante.

La communication des élèves entre eux d'une part, entre les élèves et l'enseignant d'autre part est aussi une direction importante à prolonger: dans la situation que nous observons au chapitre 10, la seule exigence est la production d'un programme dans le langage de programmation utilisée; nous avons observé que cette situation conduit dans une certaine mesure les élèves à tenter des **formulations en langage naturel**, qui apparaissent donc nécessitées par la résolution du problème, et non par des exigences didactiques; ces formulations sont rarement opérantes, mais constituent probablement des moments importants d'évolution des représentations.

4. Les structures de données dans les apprentissages

4.1 L'interaction données-algorithmes et la construction de structures de données comme problème

Nous avons vu au chapitre 1, que la recontextualisation des concepts informatiques peut s'opérer dans des stratégies de résolution des problèmes où interagissent les structures de données et les algorithmes (à la différence de stratégies fondées sur une structuration "naturelle" et par conséquent figée des données). A

l'issue de cette thèse, nous avons conscience d'avoir exploré seulement une partie très limitée des perspectives ainsi ouvertes: la remise en cause opérée par les élèves, de la codification "naturelle" par un type numérique, d'entités d'aspect extérieur numérique comme le numéro INSEE, le passage d'une codification à une autre selon le jeu de fonctions à faire opérer, montrent qu'une problématique faisant intervenir le choix d'une structure à partir d'une analyse au moins implicite du problème, peut avoir du sens y compris au niveau de débutants en difficultés.

Dans notre étude expérimentale, c'est avec la codification d'un jeu de relations entre objets, en classe de première, que le problème de la construction d'une structure de données adaptée à un traitement à partir de ce jeu de relations, se pose réellement: la situation observée permet de mettre en évidence les difficultés que les élèves rencontrent dans cette construction, mais sauf dans le cas particulier d'un élève particulièrement inventif, les élèves ne parviennent pas à élaborer de façon autonome une codification adéquate.

De fait dans les premiers apprentissages, les problèmes restant modestes, l'articulation objets-traitements est simple, et n'appelle pas d'étude épistémologique sophistiquée. **Dans ces apprentissages, la différenciation et l'articulation des éléments des S.R.T. relatifs d'une part aux données, et d'autre part aux traitements apparaissent par contre comme des processus cognitifs essentiels.** A un autre niveau, celui des études universitaires, l'étude épistémologique montre la complexité et le caractère significatif des articulations objets-traitements, par exemple dans la création d'algorithmes ([ARSAC 83], [PAIR MOHR SCHOTT 88]): le choix de la codification et de l'organisation des données interagit avec la construction du traitement dans un processus qui peut comporter plusieurs phases, une modification de la structuration des données entraînant un adaptation du traitement, ou l'inverse. L'aspect cognitif d'une telle articulation n'en est pas moins essentiel: **le passage d'une structuration des données à une autre, d'un traitement à l'autre, mettent en jeu la présence chez le sujet de Systèmes de Représentations et de Traitements** mobilisables pour chacune des étapes du processus et articulant les objets et les traitements, ainsi que la mise en relation de ces Systèmes.

4.2 L'institutionnalisation des fonctions construites en réponse à des problèmes

Nous avons indiqué que la définition d'une structure de données peut se faire à l'aide d'un jeu réduit de fonctions, les autres fonctions se construisant à l'aide de ce jeu réduit d'abord comme réponse locale à des problèmes avant d'être institutionnalisées en tant que partie intégrante de la structure; il nous semble en effet, que ce type de construction permet de mieux donner une signification à la structure, un peu comme, en mathématiques, la démonstration de théorèmes à l'aide d'un jeu de propositions initiales structure l'édifice que constitue une théorie donnée.

Nous avons utilisé avec les élèves un jeu réduit de fonctions sur les chaînes, et la fonction `position`, a été en fait construite dans de nombreux problèmes (CLASSE, CRYPTAGE, ALPHA3, ALPHA4, PIANO...) comme réponse locale. Ce choix est apparu intéressant en particulier parce qu'il a permis de développer la classe de problèmes "recherche-translation" dont nous avons rappelé ci-dessus le caractère productif. Pour autant, **les élèves n'ont pas dégagé, à partir de ces activités de résolution, la fonction `position` comme un objet précisément repéré, ni même peut-être comme un outil commun à ces situations.** La question du moment à partir duquel cette mise en évidence pourrait se faire, et donc où la dialectique outil-objet pourrait fonctionner, reste donc posée. Un aspect important de cette question est de savoir si cette mise en évidence suppose que le langage comporte la possibilité d'actions ou de fonctions nommées par le programmeur (procédure modifiant un de ses argument, fonction), ou si à un moment donné, les élèves peuvent reconnaître dans une fonction pré-définie (non introduite antérieurement), la fonction qu'ils ont construite sans la nommer. Dans ce domaine, également, il reste donc

beaucoup à faire.

5. Directions de travail pour l'élaboration de stratégies pédagogiques

Les conclusions qui précèdent permettent de dégager des directions de travail, qui, nous semblent-il, apportent de nouvelles perspectives pour un enseignement d'informatique à des débutants. Le cadre "impératif" que nous avons choisi pour l'étude expérimentale conduit à ce que ces directions s'expriment sur des notions liées à ce cadre.

- **Insérer les différents types dans les progressions visant à l'acquisition des premiers traitements.** Chaque type implique une conception des rapports entre objets et traitements. Une spécificité du type cardinal est que ce rapport est marqué par un contexte mathématique faisant l'objet d'acquisitions et de représentations inégales selon les sujets. L'utilisation de plans issus de ce contexte suppose en fait une maîtrise des objets et procédures mathématiques que beaucoup de débutants n'ont pas atteinte. Au contraire, d'autres types (chaînes, booléens, ordinaux...) peuvent permettre au programmeur débutant de s'appuyer sur des procédures familières utilisées dans des situations connues, tout en l'obligeant à construire des représentations du dispositif informatique en adaptant ces procédures et en remettant en cause certaines représentations spontanées. Il est donc nécessaire d'envisager des stratégies pédagogiques où des objets non cardinaux sont utilisés très tôt et où la construction des traitements s'opère sur ces différents objets. Il est nécessaire également de s'appuyer sur une activité autonome de l'élève, l'enseignant ne pouvant intervenir que s'il a pu prendre connaissance des représentations actuelles de l'élèves, ou fait des hypothèses à ce sujet, donc seulement à l'issue d'une période d'observation de l'activité autonome de résolution.
- **Faire utiliser l'affectation pour décomposer les calculs**: nous avons vu que des débutants éprouvent des difficultés à donner une signification aux écritures fonctionnelles composées intervenant dans la résolution de problèmes même très simples, mettant en jeu des objets non cardinaux: en fait, la composition des fonctions n'est pas maîtrisée, et leur compréhension spontanée est celle où les écritures fonctionnelles codent des actions sur les objets, dans une représentation du langage du dispositif directement déduite de l'expérience familière. Avec des élèves en difficulté, la décomposition du processus de résolution en calculs élémentaires, les résultats partiels étant affectés à des variables, contribue à une représentation où le processus de résolution est envisagé comme un calcul, et donc où les objets sont compris comme calculables.
- **Eviter la facilité de l'affichage des résultats partiels**: dans les chaînes, particulièrement, l'affichage successif de résultats partiels permet de faire l'économie de la formation du résultat comme concaténation de ces résultats partiels (par exemple, s'il s'agit de "retourner un chaîne", on pourrait faire afficher les caractères dans l'ordre inverse, ce qui "court-circuiterait" une difficulté essentielle de ce problème: l'écriture de la concaténation dans le corps de boucle). Nous avons montré qu'il est possible d'établir un contrat selon lequel tout résultat calculé doit faire l'objet d'une affectation avant son affichage, de façon à pouvoir être réutilisé dans un calcul ultérieur, et que ce contrat est positif avec des élèves en difficulté, en les obligeant à utiliser la concaténation, en les aidant à distinguer affichage et calcul, et à articuler constantes et variables.
- **Utiliser le typage des données pour une meilleure conception des objets.** Dans la perspective tracée ci-dessus, la nécessité de typer les objets par une déclaration explicite et l'utilisation de fonctions de conversion, loin d'être une contrainte arbitraire, permet une clarification des rapports entre les types, et contribue à différencier les données informatiques des objets intuitifs.
- **Rechercher des problèmes mettant réellement en jeu les types.** Nous avons vu que dans de nombreux exemples ou exercices, les réponses exactes sont *compatibles* avec des représentations erronées, et contribuent donc à les rendre

résistantes. Pour que les processus d'acquisition envisagés ci-dessus fonctionnent, il importe de vérifier que les problèmes mettent effectivement à contribution les propriétés les moins intuitives de la structure; nous avons souligné que la construction de ces problèmes suppose la connaissance par l'enseignant des représentations des élèves, ou des hypothèses sur ces représentations. D'autre part, les fonctions relatives à un type ne doivent pas être considérées de façon indifférenciée; au contraire, la donnée d'un jeu restreint permet la construction des autres fonctions en réponse à des problèmes, et structure donc l'ensemble des fonctions, en même temps qu'elle conduit à des problèmes mettant réellement en jeu la structure.

- **Maîtriser l'insertion des objets dans l'apprentissage des traitements**: les premiers traitements complexes doivent se construire sur différents type de données de façon à jouer sur l'articulation objets-traitements. Encore faut-il que l'élève parvienne à "sortir" de représentations naïves impliquant à la fois les objets et les traitements et, de ce fait, particulièrement résistantes. Pour cela, nous avons proposé d'utiliser le déséquilibre résultant de l'adaptation des représentations des données à un calcul sur un dispositif automatique, pour provoquer un réequilibrage dans le domaine des traitements, mais nous avons constaté que ceci est possible seulement si l'élève a commencé à adapter ses représentations des objets. Dans cet esprit, des exercices simples mettant en jeu l'affectation, la notation fonctionnelle, les différentes fonctions d'un jeu restreint semblent indispensables avant d'aborder les traitements complexes (itération, alternatives complexes). Là aussi, la prise de connaissance par l'enseignant des représentations des élèves à partir de l'observation de leurs productions est essentielle.

6. Pour conclure

Nous avons mené cette thèse à partir d'un double questionnement. Quelle place donner aux objets dans un enseignement de l'informatique destiné à des débutants? Quelles difficultés précises rencontrent les débutants dans l'apprentissage de la programmation? Cette problématique nous a permis à la fois une meilleur compréhension des processus de résolution et d'acquisition des débutants, et de proposer des directions de travail nouvelles pour la définition de cursus destinés à ces mêmes débutants. Elle permet également que de nombreuses questions soient ouvertes. Elles s'est donc révélée tout à fait productive.

BIBLIOGRAPHIE

ANDERSON J.R. FARELL R. SAUERS R. (1984) «Learning to program in LISP» *Cognitive Science 8*

ARSAC J. (1980) *Premières leçons de programmation* CEDIC

ARSAC J. (1983) *Les bases de la programmation* DUNOD

ARSAC J. (1986] «L'informatique et le sens. Une gigantesque mutation culturelle ?» *Actes du colloque du CREIS histoire et épistémologie de l'informatique*

ARSAC J. (1990) «Vous avez dit algorithmique» *Actes du 2ème colloque francophone sur la didactique de l'informatique* Presses Universitaires de Namur

ARTIGUES M. (1991) «Epistémologie et Didactique» *Recherche en didactique des Mathématiques*, Vol 10/2.3. Editions La Pensée Sauvage

AUTHIER A. , WAITER N. (1989) «Modèles, objets et pédagogies de l'option informatique» *INFORMATIQUES* n°6 Publié par le Ministère de l'Education nationale, CRDP Poitiers.

BARNES J.G.P (1980) «An overview of Ada» *Software practice & experience*, vol. 10 1980

BARON G.L. (1989) «L'option informatique à la rentrée de 1988/89» *INFORMATIQUES* n°6 Publié par le Ministère de l'Education nationale, CRDP Poitiers.

BARON G.L.(1988) *L'informatique comme discipline scolaire.* PUF.

BENETOLLO R. (1987) «Place dans l'option informatique de langages tels que LOGO, PROLOG et LISP» *Pratiques et Savoir-faire des élèves de l'option informatique.* Publié par le Ministère de l'Education nationale, CRDP Poitiers.

BENETOLLO R. (1990) *Hétérogénéité et réussite dans les classes d'option informatique des lycées* Document distribué au 2ème colloque francophone sur la didactique de l'informatique (Namur).

BOURBION M. et al. (1984) *L'alternative LOGO* Armand Colin-Bourrelier.

BOUSSARD J.C. MAHL R. (1985) *Programmation avancée* Eyrolles

BROUSSEAU G. (1986) «Fondements et méthodes de la didactique des Mathématiques» *Recherche en didactique des Mathématiques*, Vol 7.2. Editions La Pensée Sauvage.

CANAL M.(1983) *Parler LSE et apprendre à l'utiliser* Eyrolles

CAPPONI B. BALACHEFF N. (1989) «Tableur et Calcul Algébrique» *Educational Studies in Mathématics* n°20 Kluwer Academic Publishers

CASANOVA G. (1967) *L'Algèbre de Boole* PUF

CHAMBADAL L. (1983) *Calcul Pratique* HACHETTE

CHEVALLARD Y. (1985) *La transposition didactique* La pensée sauvage

CLOUTIER J.F. (1988) «Apports de différents paradigmes de programmation comme autant d'outils de pensée» *Actes du premier colloque francophone de didactique de l'informatique* Editions EPI

CNDP (1987) *OPTION INFORMATIQUE Classes de seconde, première et terminale.*CNDP

COHORS-FRESENBORG E. (1987) «The Benefit of Microwords in learning computer programming» *Osnabrücker Schriften zur Mathematik* 04/87

CURFIP-USTLFA (1989) *Compte-rendu des travaux des enseignants de l'Option Informatique* CURFIP-USTLFA

DAGDILELIS V. BALACHEFF N. CAPPONI B. (1991) «L'apprentissage de l'itération dans deux environnements informatiques» *ASTER* n°11 INRP.

DIJKSTRA E.(1976) *A discipline of programming* Prentice Hall

DOUADY R. (1986) Jeux de cadres et dialectique outil-objet" *Recherches en didactique des Mathématiques*, Vol 7.2. Editions La Pensée Sauvage

DUCHATEAU Ch. (1991) «Synthèse d'un colloque en trois temps..» *Actes du Second Colloque Francophone de Didactique de l'Informatique* Presses Universitaires de Namur

DUCRIN A. (1984) *Programmation 1* Dunod

DUFOURD G. DUFOURD J.F.(1988) «Des univers et des outils variés pour commencer à programmer» *Actes du Premier Colloque Francophone de Didactique de l'Informatique* Editions EPI

du BOULAY B. (1988) «Intelligent Systems for Teaching Programming» *IFIP, Artificial Intelligence Tools in education.* Elsevier Science

ERNST G.W. NEWEL A. (1969) *GPS: A case study in generality and problem solving* London Press

FAVRE-NICOLIN R. (1988) «Le tableur et l'option informatique-vers la programmation par objet» *Actes du premier colloque francophone de didactique de l'informatique* Editions EPI

GARLAND S.J. (1973) *BASIC a specification* Dartmouth College HANOVER New Hampshire

GATES B. (1989) «The 25th Birthday of BASIC» *Byte* (octobre 1989).

GAUDEL M.C. SORIA M. FROIDEVAUX Ch. (1986] *Types de données et algorithmes* INRIA

GEOFFROY C. (1988) «Une initiation à l'analyse descendante» *Actes du premier colloque de didactique de l'Informatique* Editions EPI

GIANESINI H. et al. (1985) *Prolog* InterEditions

GREEN T.R.G. (1977)«Conditional program statements and their comprehensibility to professionnal programmers» *J.occup.Psychol* n°50

GRISWOLD R. (1975) *String and list processing in SNOBOL4* Prentice-Hall

HEUZE F. (1990) «Une démarche pour l'apprentissage de la programmation en classe de seconde» *Actes du Second Colloque Francophone de Didactique de l'Informatique* Presses Universitaires de Namur

HILLEL J. (1986) «Mathematical and Programming concept in a restricted LOGO environnement» *Recherches en didactique des Mathématiques*, Vol 6/2.3 Editions la Pensée Sauvage

HOC J.M. (1977) «Role of mental representation in learning a programming language» *Int. Man-Machine Studies* n°9

HOC J.M. (1987) *Psychologie cognitive de la planification* Editions PUG

HOFSTADTER D. (1979) *Bach Echer Gödel* Basic Books (paru en traduction française InterEditions 1985)

HOROWITZ E. (1983) *Fundamentals of programming languages* Springer Verlag

JENSEN K. WIRTH N. (1978) *PASCAL User Manual and Report* Springer Verlag (Traduction française: *Pascal, manuel de l'utilisateur* Eyrolles 1980)

KNUTH D. (1967) *The art of computer programming Tome 1* Addison-Wesley

LABORDE C., BALACHEFF N., MEIJAS B. (1985) «Problématique et genèse du concept d'itération, une approche expérimentale» *ENFANCE* 2-3

LELOUCHE R. MALOUIN J. (1991) «Fondements pédagogiques et éléments de design d'un système d'E.A.I.O. de l'algorithmique» *Actes du Second Colloque Francophone de Didactique de l'Informatique* Presses Universitaires de Namur

LOGO Manuel *LOGO-Manuel de Référence* CEDIC-Nathan 1985

LSE 8086/8088 *Manuel d'utilisation* Microdur 1984

LUCAS M. (1983) *Algorithmique et représentation des données Tome 2* MASSON

LUCCI A. (1989) *Production de logiciels pour l'enseignement: une expérience de prototypage d'un système construit sur un environnement Prolog* Thèse de 3ème cycle INP Grenoble

M.E.N. 1991 *Quel lycée pour demain* éditions CNDP-Le Livre de Poche

MENDELSOHN P. (1987) «Apprentissage d'un système de commande informatisé et organisation des connaissances chez l'enfant» *Psychologie Française* n°4

MEYER B. BAUDOIN C. (1978) *Méthodes de programmation* Eyrolles

MILLOT D. 1988 Introduction aux langages orientés objets *Bulletin de l'association E.P.I. n°52*

MORVAN P. et alt. (1981) *Dictionnaire de l'informatique* Larousse

MULLER P. (1987) «La désignation des personnages dans le tartuffe de Molière» *Bulletin de l'association E.P.I. n°47*

NOYELLES Y. (1989) «La saga du LSE et et de sa famille» *Bulletin de l'association E.P.I. n°54*

PAIR C. (1988) «L'apprentissage de la programmation» *Actes du premier colloque francophone de didactique de l'informatique* Editions EPI

PAIR C. MOHR R. SCHOTT R. (1988) *Construire les algorithmes* Dunod

PAPERT S. (1980) *MINDSTORMS* Basic Books (Traduit en français sous le titre de *Jaillissement de l'esprit* Flammarion 1981).

PEA R.D. (1984) «Language-independant conceptual "bugs" in *novice programming*» *Journal of Educational Computing Research* 1-12

PIAGET J. SZEMINSKA A. (1967) *La genèse du nombre chez l'enfant* Delachaux Niestlé

RICHARD C. RICHARD D. (1985) *Programmatique* Belin

ROGALSKI J. (1984) *Sur une séquence didactique en informatique* ; rapport de recherche IMAG n°50

ROGALSKI J. (1987) *Acquisition de savoirs et de savoir-faire en informatique* IREM Paris VII

ROGALSKI J. (1988) «Acquisition de structures conditionnelles » *Annales de Didactique et de Sciences cognitives n°1* IREM de Strasbourg 1988)

ROGALSKI J. (1991) «Approches cognitives de la programmation » *Actes du Second Colloque Francophone de Didactique de l'Informatique* Presses Universitaires de Namur

ROY J.P. (1990) «La programmation fonctionnelle» *Informatiques n°7* Publié par le Ministère de l'Education nationale, CRDP Poitiers.

SAMMET J.E. (1969) *Programming languages; History & Fondamentals* Prentice-Hall

SAMURCAY R. (1985) «Signification et fonctionnement du concept de variable informatique chez des élèves débutants» *Educational Studies in Mathematics* n°16 Kluwer Academic Publishers

SAMURCAY R. (1987) «Plans et schémas de programme» *Psychologie Française* n°4

SAMURCAY R. ROUCHIER A. (1990) «Apprentissage de l'écriture et de l'interprétation des procédures récursives» *Recherches en didactique des Mathématiques*, Vol 10 2.3. Editions la Pensée Sauvage.

THALMAN D. LEVRAT B. (1979) *Conception et implantation de langages de programmation* Gaëtan Morin

VIVET M. (1984) «CAMELIA A knowledge based mathematical system» *ECAI.84 Advances in Artificial Intelligence* North-Holland

VON NEUMAN J. (1947) «First draft on a report on the EDVAC» Reproduit dans *Actes du colloque du CREIS histoire et épistémologie de l'informatique* Mai 1988

WEIZENBAUM J. (1976) *Computer Power and Human Reason* FREEMAN and Co. (traduit en Français sous le titre *Puissance de l'Ordinateur et Raisonnement de l'Homme* Editions d'Informatique 1981).

WIRTH N. (1976) *Algorithms + Data Structures = Programs* Prentice Hall