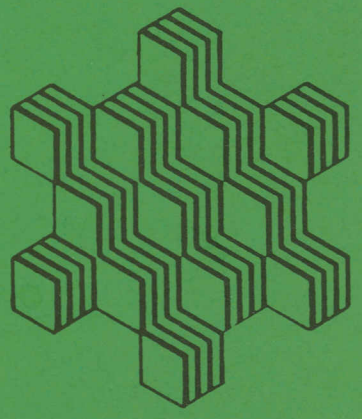
16-0081-01

GUIDE PÉDAGOGIQUE

• Secondaire

INTRODUCTION À LA SCIENCE DE L'INFORMATIQUE



Québec

INTRODUCTION À LA SCIENCE DE L'INFORMATIQUE

Direction générale du développement pédagogique
Direction de la formation générale

**Approuvé par les Comités protestant et catholique
du Conseil supérieur de l'éducation
les 23 septembre et 20, 21 octobre 1983.**

ISBN 2-550-06294-9

Dépôt légal - troisième trimestre 1984

TABLE DES MATIÈRES

Introduction

Introduction

1 - Rôle du guide pédagogique

Le guide se propose d'indiquer quels sont les objectifs prioritaires du cours, comment ces objectifs peuvent être intégrés à travers l'activité de l'élève. Il suggérera des approches possibles pour le professeur, des techniques d'intervention et des méthodes de cheminement.

2 - Esprit général du programme

Le programme se veut une « introduction à la science de l'informatique ». Il s'agit d'une introduction, donc d'une première approche, qui sera par la suite approfondie. Le programme mentionne un grand nombre d'objectifs. Atteindre ces objectifs dans le cadre d'un cours d'introduction veut dire aborder les notions correspondantes de façon à pouvoir les approfondir efficacement dans l'avenir. D'autre part, on parle d'une science. Qui dit science parle de méthodes particulières et de concepts, et non pas de techniques d'utilisation d'une machine ou de connaissances reliées exclusivement au vocabulaire d'un langage de programmation. Finalement, on veut aborder le problème de la programmation dans sa dimension la plus générale qui part d'un projet à réaliser devant être mené à terme, c'est-à-dire, à un logiciel fonctionnant effectivement sur un ordinateur.

Enfin ce cours s'adresse à la clientèle la plus vaste possible, donc à des élèves qui ne poursuivront peut-être pas des études dans le domaine de l'informatique. Ceci impose une approche accessible au plus grand nombre et la poursuite d'objectifs de formation générale. À ce propos le programme *Introduction à la science de l'informatique* désire augmenter les aptitudes à la résolution de problèmes tout en améliorant l'autonomie intellectuelle.

I
La science de l'informatique

I - La science de l'informatique

Le programme actuel fait suite à un programme expérimental: Initiation à un langage de programmation. Le changement de nom correspond à l'évolution des techniques de programmation elles-mêmes — les premiers cours étaient souvent des cours de langages où l'on apprenait des mots. Le langage le plus souvent utilisé, le BASIC, ne favorisait pas non plus l'acquisition de concepts conduisant à des programmes clairs, fiables, construits méthodiquement.

Depuis, les choses ont évolué. Le BASIC est toujours le langage le plus répandu, du moins comme langage d'initiation. Par contre, les programmes se font de façon structurée et des techniques d'analyse descendante se sont développées. On sait qu'il est possible d'écrire des programmes lisibles, même en BASIC. Les techniques de résolution de problèmes sont devenues si importantes qu'on les introduit explicitement dans les cours de mathématiques. La programmation devient de plus en plus une science, avec ses méthodes, ses concepts, ses modes de représentation, le tout étant le plus souvent indépendant d'un langage particulier. La « science » de l'informatique permet maintenant de passer facilement d'un langage à l'autre. Ceci ne veut pas dire que tous les langages sont équivalents, mais l'initiation ne doit pas se faire à un langage mais à un ensemble de concepts et de méthodes qui s'incarnent à travers l'utilisation que l'on fait du langage; ceci se traduit aussi dans le comportement face à un problème à solutionner par la programmation.

1. Le comportement sauvage ou spontané

- Devant le problème à résoudre, le premier réflexe est de commencer à programmer sur l'ordinateur.
- Par une succession d'essais et d'erreurs on se rapproche de la solution et quelquefois un plan se dégage peu à peu.
- Le plus souvent les corrections deviennent de plus en plus difficiles; le programme devient illisible. Au bout d'un certain temps, il faudra sans doute tout recommencer.
- Les lacunes d'une telle approche sont évidentes. Cependant, il ne faut pas négliger certains acquis venant de ce comportement « sauvage ».

Par exemple, l'enthousiasme; c'est l'atmosphère du Club! Et puis certains jeunes placés dans cette situation ont fait montre d'initiative, d'imagination et ont réalisé des choses étonnantes. Pour eux, ce fut souvent l'occasion de réaliser réellement une oeuvre personnelle dont il pouvait être fière. Par contre, l'absence de méthode et la méconnaissance de certains concepts fondamentaux font que le plus grand nombre se décourage.

2. Le comportement souhaité

- À l'énoncé du problème à résoudre, le premier réflexe est d'en préciser les données et les résultats escomptés.
- Pour faire le lien entre données et résultats, on invente un algorithme en utilisant certaines techniques de résolution de problèmes.
- À partir de là, on construit un plan de futur programme: les diverses parties sont bien dégagées et les liens entre elles précisés. Tout cela peut être représenté graphiquement — c'est un travail d'architecte.

- La rédaction du programme va pouvoir commencer. Chaque section est d'abord décrite en pseudo-langage, facile à lire. Il reste ensuite à traduire dans le langage utilisé et à documenter chaque morceau de programme.

- Le programme est ensuite testé et corrigé; ce qui est facile parce que c'est un programme structuré et documenté.

- Enfin le programme devra sans doute être modifié pour être adapté à des besoins nouveaux. Il sera possible d'en utiliser des parties, ou d'en intercaller de nouvelles. La structure du programme est telle que ceci est possible sans que sa cohérence ne soit affectée.

3. Le défi pédagogique

Il s'agit, partant du « comportement sauvage », d'amener les élèves à avoir un comportement plus près de l'attitude que l'on vient de décrire, tout en conservant l'enthousiasme initial. Le défi est de taille, et il faut pour cela développer une méthodologie particulière. Cependant, avant d'aborder les problèmes de méthodologie, il faut préciser la méthode de travail utilisée en programmation et les concepts particuliers que l'on veut développer, et c'est pourquoi l'on parlera d'abord des méthodes et des concepts propres à la science de l'informatique. Dans la deuxième partie, l'on abordera les problèmes pédagogiques.

4. Méthodes et concepts

4.1 Attaquer le problème général

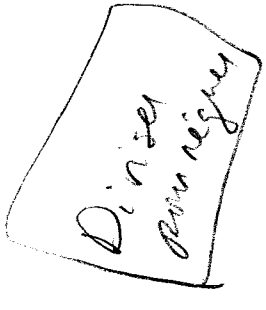
- En précisant les données et les inconnues.

- En spécifiant le but recherché et les résultats anticipés.

- En reformulant le problème et en utilisant des schémas, des graphes, des tableaux pour mettre en évidence sa structure.

4.2 Décomposer le problème en sous-problèmes plus simples

- En identifiant plusieurs étapes conduisant à la résolution du problème (problèmes auxiliaires).



- En cherchant un sous-problème pouvant servir à la résolution de plusieurs autres sous-problèmes.

- En transformant le problème de départ en un problème moins général, plus facile à résoudre.

- En prenant un cas particulier, ou un cas limité du problème général.

- En cherchant les problèmes auxiliaires que l'on sait déjà résoudre.

- En faisant une hypothèse de récurrence.

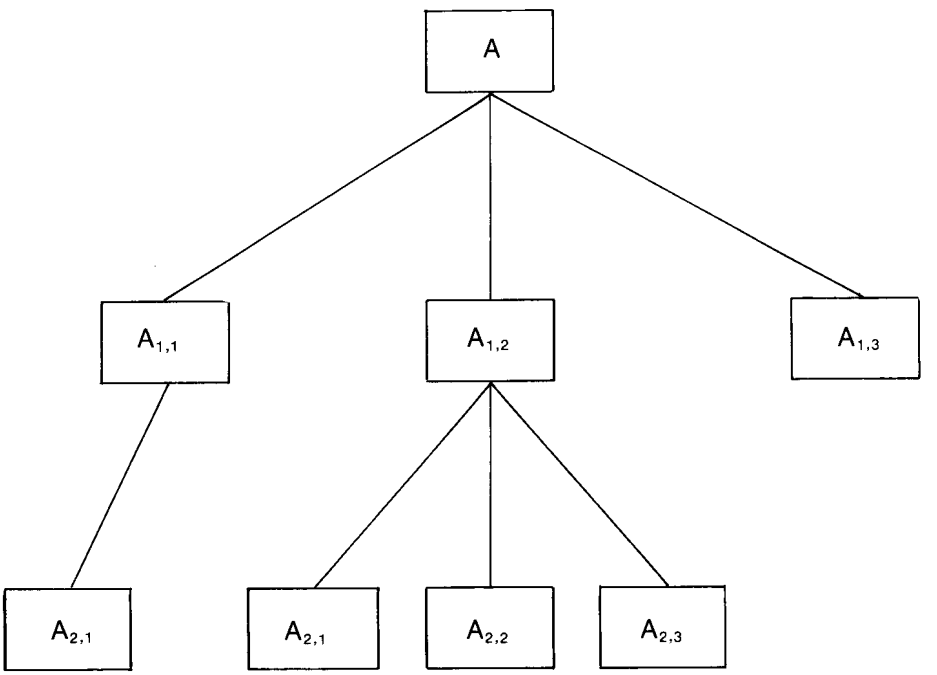
4.3 Recomposer une tâche à partir de plusieurs tâches élémentaires

Souvent, on se demande comment résoudre le problème général à partir d'un problème plus élémentaire que l'on sait résoudre; soit que l'on répète le problème élémentaire, soit que l'on utilise l'itération (1) ou la récursivité (1). Ainsi certaines constantes sont alors considérées comme des variables et l'on a à généraliser ce qui devient un cas particulier.

4.4 Le plan: quelques techniques

Cette décomposition du problème principal débouche sur un plan d'action qui conduira finalement au programme proprement dit.

Le plan sera présenté le plus souvent sous forme graphique: on y distinguera clairement un certain nombre de tâches à accomplir et reliées entre elles. On obtiendra des graphes de ce type:



Pour réaliser la tâche A, il faut réaliser les tâches $A_{1,1}$, $A_{1,2}$, $A_{1,3}$.

Pour réaliser la tâche $A_{1,1}$, il faut réaliser la tâche $A_{2,1}$.

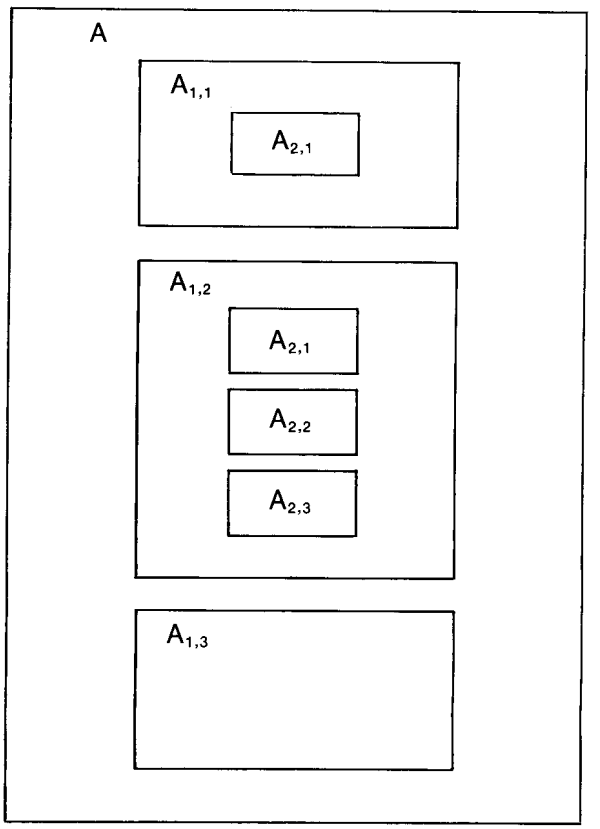
Pour réaliser la tâche $A_{1,2}$, il faut réaliser les tâches $A_{2,1}$, $A_{2,2}$, $A_{2,3}$.

La tâche $A_{1,3}$ ne demande la réalisation d'aucune autre tâche.

La tâche $A_{2,1}$, utilisée par $A_{1,1}$ et $A_{1,2}$, pourra être réalisée par un sous-programme ou une procédure.

(1) voir I, 4.10

On peut aussi représenter l'ensemble des tâches à accomplir de cette façon:



On doit obtenir une structure « emboîtée », analogue à une structure parenthèsée en algèbre. Il faut veiller à ce qu'une tâche soit bien effectuée à l'intérieur d'une autre tâche, et non pas réalisée partiellement dans plusieurs tâches.

4.5 Décrire chaque tâche en pseudo-langage

Lorsque le déblayage précédent est terminé, on peut alors décrire chaque tâche à effectuer un peu comme on indique une recette dans un livre de cuisine: verser les oeufs dans la casserole, faire chauffer à feu doux, ajouter un demi-litre d'eau. . ., etc. On a une suite d'énoncés, correspondant à des actions plus ou moins complexes qui indiquent cependant une suite d'opérations à réaliser. Chaque opération peut être décomposée en une suite d'actions plus élémentaires; là encore, on peut se référer au modèle culinaire. Faire une omelette peut se décomposer en casser les oeufs, battre les oeufs, verser les oeufs dans la poêle, etc. Le pseudo-langage permet une méthodologie qui permet finalement d'obtenir un algorithme programmable (codable) dans le langage de programmation utilisé.

Le pseudo-langage est structuré pour le « et » logique, le « ou » logique, « aller à », « répéter. . . jusqu'à » ou « tant que. . . faire », « si. . . alors ». On y ajoute l'affectation (←) « entrer » pour introduire une valeur et « afficher ».

On peut y ajouter d'autres structures, comme la boucle « Pour: Pour I variant de 0 à M, répéter. . . », la structure « Au cas où ». Il faut cependant se limiter en fonction des objectifs que l'on veut atteindre. Naturellement, le pseudo-langage n'est pas un langage de programmation. C'est un instrument d'analyse. On l'appelle aussi le pseudo-code, ou le langage A (A pour algorithme).

EXEMPLE: TRACER UN TRIANGLE ISOCÈLE DANS LA POSITION INDIQUÉE PAR LE DESSIN, EN UTILISANT LE MODE TEXTE DE L'ORDINATEUR. LA DIMENSION DU TRIANGLE EST VARIABLE.

A

A A

A A A

A A A A

A A A

A A

A

Écriture du programme en pseudo-langage

- Longueur du triangle: 2L + 1
  compteur: n
- Entrer L
- n ← 1
- Répéter

  Tracer une ligne de A de longueur n
  Passer à la ligne suivante
  Augmenter n de 1

  Jusqu'à ce que n = L + 1
- n ← L
- Répéter

  Tracer une ligne de A de longueur n
  Passer à la ligne suivante
  Diminuer n de 1

  Jusqu'à ce que n = 1

On peut maintenant décomposer la partie:

Tracer une ligne de A de longueur n

```
Compteur: I
I ← 1
Répéter
   Placer un « A » à la position I sur la ligne
   Augmenter I de 1
Jusqu'à ce que I = n
```

On obtient alors le programme suivant en pseudo-langage:

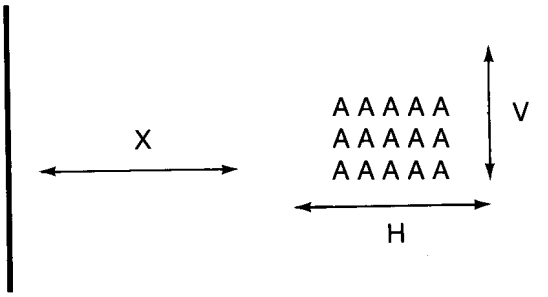
```
• Longueur du triangle: 2L + 1
  Compteurs: n et I
• Entrer L
• n ← 1
• Répéter
     I ← 1
     Répéter
        Placer un « A » à la position I sur la ligne
        Augmenter I de 1
     Jusqu'à ce que I = n
     Passer à la ligne suivante
     Augmenter n de 1
  Jusqu'à ce que n = L + 1
• n ← L
• Répéter
     I ← 1
     Répéter
        Placer un « A » à la position I sur la ligne
        Augmenter I de 1
     Jusqu'à ce que I = n
     Passer à la ligne suivante
     Diminuer n de 1
  Jusqu'à ce que n = 1
```

L'utilisation d'un pseudo-langage permet de voir immédiatement la structure du programme; il est lisible par tout le monde, et peut être écrit indépendamment du langage de programmation. C'est un outil d'analyse et un moyen de conceptualisation.

4.6 Rédaction du programme: variable locale, variable globale, paramètre

Une fois que le programme à écrire est résumé sous la forme d'un pseudo-langage, il s'agit de rédiger de petits programmes reliés entre eux.

Exemple: Tracer un rectangle plein composé de H caractères « A » horizontaux et de V caractères « A » verticaux. Le rectangle est situé à X positions du bord de l'écran:



Programme principal en pseudo-langage

```
Entrer V
Entrer H
Entrer X
I ← 1

Répéter
   Tracer une ligne (X, X + H)
   Passer à la ligne suivante
   Augmenter I de 1
Jusqu'à ce que I = V
```

Il reste à faire l'unité de programme: Tracer une ligne (X, Y)

Version: 1

```
Répéter
   Afficher le caractère « A » à la position X
   Augmenter X de 1
Jusqu'à ce que X = Y
```

Cette partie de programme peut constituer une unité, une procédure, ou un sous-programme, selon le désir du programmeur et le langage utilisé. Le programme principal devrait fonctionner. L'unité de programme fait ce que l'on attend d'elle: elle trace une ligne de la position X à la position Y.

On pourrait en déduire que le programme suivant fonctionne:

```
Entrer V
Entrer H
Entrer X

Y ← X + H
I ← 1

Répéter
  Répéter
    Afficher le caractère « A » à la position X
    Augmenter X de 1
  Jusqu'à ce que X = Y
  Passer à la ligne suivante
  Augmenter I de 1
Jusqu'à ce que I = V
```

Il n'en est rien: la valeur de X est modifiée dans la deuxième boucle. Or, cette valeur doit demeurer constante pendant toute la durée du programme: X est une variable globale. Sa valeur doit pouvoir être utilisée n'importe où dans le programme. On ne doit pas la modifier pendant l'exécution.

Corrigeons maintenant l'unité de programme de la façon suivante:

Version: 2

```
I ← 0

Répéter
  Afficher le caractère « A » à la position X + 1
  Augmenter I de 1
Jusqu'à ce que I = H
```

Utilisé seul, ce programme fonctionne. Inséré dans le programme principal, il est possible que ce programme ne fonctionne pas.

```
Entrer V
Entrer H
Entrer X

Y ← X + H
I ← 1

Répéter
  I ← 0

  Répéter
    Afficher le caractère « A » à la position X + 1
    Augmenter I de 1
  Jusqu'à ce que I = H
  Passer à la ligne suivante
  Augmenter I de 1
Jusqu'à ce que I = V
```

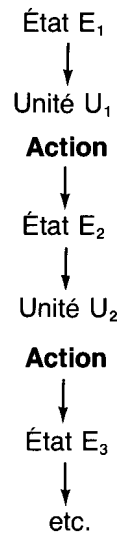
Dans l'unité de programme (version: 2), le I est utilisé pour la gestion interne du programme. Sa valeur après l'exécution du programme n'a aucun intérêt pour le reste du programme. Cette variable est *locale* à ce programme. Certains langages permettent de la déclarer « locale » et il n'y a pas de difficulté; d'autres ne le permettent pas. Il faut alors, par exemple, utiliser des variables différentes dans chaque unité de programme.

On peut écrire maintenant le programme de la façon suivante:

```
Entrer V
Entrer H
Entrer X

Y ← X + H
I ← 1

Répéter
  J ← 0

  Répéter
    Afficher, le caractère « A » à la position X + 1
    Augmenter J de 1
  Jusqu'à ce que J = H
  Passer à la ligne suivante
  Augmenter I de 1

Jusqu'à ce que I = V
```

4.7 Qu'est-ce qu'un programme fait ou Qu'est-ce qu'un programme change? Notion d'état

Dans l'exemple précédent, « l'unité de programme » faisait ce qu'on attendait d'elle: elle traçait une ligne de « A » partant de la position X à la position Y; et pourtant le programme principal ne fonctionnait pas parce que l'unité de programme, en plus de *faire* ce qu'on attendait d'elle, *changeait* aussi les valeurs de la variable globale X dans la première version, ou de la variable I dans la seconde version. Il ne faut donc pas se demander ce qu'un programme fait mais en quoi il change l'État E_1 avant son exécution en État E_2 après son exécution, un état étant défini par l'état de l'ensemble des variables à un moment donné de l'exécution. Le déroulement d'un programme se caractérise par une suite d'évolution d'états E_1, E_2, . . ., chaque « unité de programme » assurant le passage d'un état à un autre.



Il importe donc de contrôler cette suite d'états E_1, . . . , E_n le plus précisément possible.

4.8 Sous-programmes, procédures

Reprenons le programme principal utilisé dans la partie précédente.

```
Entrer V
Entrer H
Entrer X

I ← 1

Répéter
   Tracer une ligne (X, X + H)
   Passer à la ligne suivante
   Augmenter I de 1
Jusqu'à ce que I = V
```

Quelque soit le langage utilisé, la phrase « Tracer une ligne (X, X + H) » ne correspond à aucune instruction. On peut alors construire un sous-programme, ou une procédure, que l'on pourra appeler chaque fois que cela est nécessaire, et qui réalisera: « Tracer une ligne (X, X + H) ». On place les procédures ou les sous-programmes au début, ou à la fin, l'important étant d'isoler le programme principal pour pouvoir se concentrer sur sa structure.

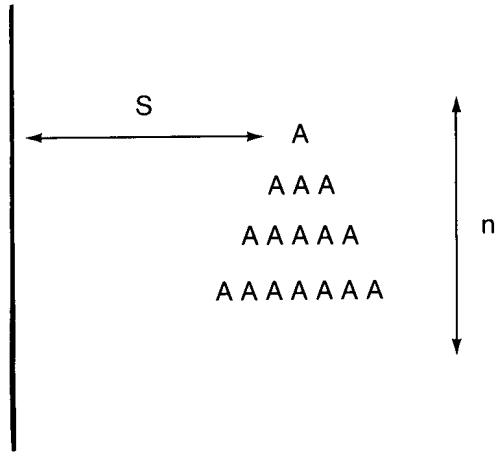
On rédige le programme principal en « pseudo-langage » sans avoir à s'occuper de la définition des sous-programmes. On débouche ainsi sur une structure modulaire, facile à lire, donc à corriger et à contrôler.

Nous avons vu que pour pouvoir utiliser ces modules sans problème, il fallait être très prudent quant à la nature des variables. Un sous-programme peut être appelé à plusieurs reprises par le programme principal. Il faut donc le documenter avec beaucoup de précision.

4.9 Itération et récursivité

Un certain nombre de problèmes peuvent se résoudre en utilisant le raisonnement par récurrence. Prenons un exemple:

FAIRE UN PROGRAMME PERMETTANT DE CONSTRUIRE UN TRIANGLE ISOCÈLE DU GENRE DE CELUI-CI



Le sommet du triangle est à S espaces du bord de l'écran et il comporte n lignes de A.

Nous supposons que nous avons à notre disposition un sous-programme ou une procédure L (X, Y) qui trace une ligne de A commençant à la position X et se terminant à la position Y.

On doit d'abord chercher une hypothèse de récurrence: Si T_n est un triangle de hauteur n, on obtient T_n à partir de T_{n-1} et en lui ajoutant une ligne supplémentaire de A.

On obtient un programme comme celui-ci.

```
* Position de départ *
  X ← S
  Y ← S
  L (X, Y)
* Hypothèse de récurrence *
  i ← 0
  Répéter
    X ← X − 1
    Y ← Y − 1
    L (X, Y)
    Augmenter i de 1
  Jusqu'à ce que i = n
```

On retrouve la structure classique d'un raisonnement par récurrence. L'ordinateur construit le triangle, ligne à ligne, et s'arrête lorsque le processus est terminé. On vient de construire un programme itératif.

Si le langage le permet, on peut écrire le programme d'une façon totalement différente en utilisant la récursivité. Un langage récursif est un langage qui permet d'utiliser une procédure dans sa définition même. Voici un exemple qui va permettre de résoudre le problème précédent: Soit T_n la procédure qui permet de construire un triangle de hauteur n.

Pour construire T_n, on peut construire T_{n-1} puis lui adjoindre une ligne L (X − n, X + n); T_1 est obtenu grâce à L (S, S). On obtient donc le programme suivant:

```
Si n = 0 alors L (S, S)
Sinon
  T(n-1)
  L (S − n, S + n)
```

On voit la différence de longueur et de complexité de ces deux programmes. Le premier repose sur les notions d'affectation et de répétition. Le second n'a besoin d'aucune de ces deux notions. Théoriquement, un langage récursif peut ignorer la notion de structure de répétition tant que l'espace mémoire le permet, car la récursivité demande beaucoup plus de place que la répétition. On peut donc imaginer deux approches différentes au cours; l'une partant des notions d'affectation et de répétition, l'autre de la notion de récursivité. Pour cette approche, le langage BASIC ne convient pas, puisque sa principale faiblesse provient du fait qu'il ne possède pas la récursi-

vité. Il faut voir que l'approche récursive est totalement différente de l'approche itérative: un programme récursif se contente de l'écriture d'une définition (le triangle T_n est obtenu par l'adjonction d'une ligne au triangle T_{n-1}) alors qu'un programme itératif décrit pas à pas la construction du triangle (d'où son nom). Un programme récursif ignore la façon dont l'ordinateur réalise les choses, ce que décrit par contre un programme itératif. C'est peut-être ce qui fait que ces deux approches semblent si imperméables l'une de l'autre. Quelqu'un formé pour la réalisation de programmes descriptifs aura du mal tout à coup à travailler à l'aide de définitions presque mathématiques. Par contre, l'approche uniquement récursive semble demander plus de doigté pédagogique.

Le raisonnement par récurrence est difficile. Par contre, l'itération et la récursivité semblent beaucoup plus abordables pour les élèves, à la grande surprise de nombreux professeurs.

4.10 La documentation d'un programme

Chaque sous-programme, procédure, ou unité de programme doit fournir les renseignements suivants:

- Un nom caractéristique de la tâche accomplie.
- Une description brève de cette tâche.
- La liste des variables; le nom de chaque variable est choisi en fonction de ce qu'elle représente. Sa signification doit être indiquée et sa nature précisée (locale, globale, paramétrique).
- La liste des sous-programmes ou procédures nécessaires au fonctionnement de l'unité définie.

Exemple:

```
* Ce S/S programme permet de tracer
* une ligne de A
* X indique la position du début de la
* ligne
* Y indique la position du dernier
* caractère de la ligne
* Paramètres: X, Y
* Variable locale: I
* Début
  I ← 0
  Répéter
      Afficher le caractère « A » à la position X + 1
      Augmenter I de 1
  Jusqu'à ce que I = H
* Fin.
```

Cette documentation doit être effectuée au fur et à mesure que chaque unité de programme est écrite. On séparera toutes les unités de programme par des blancs et l'on veillera à travailler sur des unités de programme courtes, dépassant rarement une page de l'écran d'affichage.

4.11 Tester le programme

On commence ensuite à tester le programme par les unités élémentaires qui ne font appel à aucune autre unité de programme. Ensuite, on peut tester les unités de programme qui utilisent les unités déjà testées et l'on remonte ainsi jusqu'au programme principal.

Les stratégies pour tester un programme sont souvent équivalentes à celles qui permettent de l'écrire: ce n'est pas parce qu'un programme marche dans certains cas qu'il est correct. Il faut trouver les cas particuliers qui peuvent révéler des erreurs. Une erreur étant constatée, il faut trouver l'erreur dans la structure du programme qui l'a entraînée et non pas essayer d'en atténuer les effets en ajoutant à chaud des morceaux de programme qui vont rendre l'ensemble illisible, et bien souvent engendrer d'autres erreurs.

4.12 Résumé

Voici un résumé de travail conduisant à la rédaction d'un programme. Les mots ou expressions soulignés correspondent à des concepts qui doivent se dégager d'un cours de programmation. La « science de l'informatique » se caractérise plus par ses concepts que par le vocabulaire particulier à un langage.

- Préciser le problème à résoudre en décrivant les *données* du problème et les *résultats* attendus.
- *Décomposer l'action globale* que doit réaliser le programme en actions plus élémentaires (*démarche descendante*).
- Décrire le fonctionnement de chaque partie en utilisant un *pseudo-code lisible* par tout le monde.
- Décider des variables à utiliser en choisissant leur nom de façon imagée et en distinguant leur nature (*globale, locale, paramétrique,* . . .).
- Rédiger chaque partie dans le langage choisi en lui *donnant un titre correspondant à l'action attendue* et en *indiquant le sens et la nature de chaque variable*. On commence par les programmes élémentaires et en remontant (*programmation ascendante*): on couvre ainsi l'ensemble.
- Indiquer en quoi l'exécution d'une partie de programme qui vient d'être écrite modifie *l'état global du programme*. Cet état est constitué des valeurs, des variables, des paramètres et des constantes utilisés dans le programme complet à un instant donné.
- *Tester* chaque partie au fur et à mesure en commençant par les parties ne faisant appel à aucun autre programme (*programmes élémentaires*).

II
L'approche pédagogique

II - L'approche pédagogique

1. Considérations générales

Nous venons de voir qu'une introduction à la science de l'informatique est d'abord une méthode de travail mettant en oeuvre des concepts particuliers. Les acquisitions devront donc se manifester bien plus au niveau des attitudes et des démarches utilisées qu'au niveau de la connaissance du vocabulaire relié à un langage de programmation spécifique.

Programmer est une activité globale, partant de la définition d'un problème et se terminant par un programme tournant effectivement sur un ordinateur particulier. L'ensemble de ces composantes doit donc apparaître dans l'apprentissage, et c'est pourquoi le programme privilégie « l'approche par projet ». Celle-ci permet, d'autre part, de partir de l'enthousiasme souvent naturel des élèves pour tout ce qui touche l'informatique pour leur permettre de structurer leur démarche et leur pensée. Elle permet aussi aux élèves de réaliser des « choses qui leur ressemblent », donc de les rendre plus créatifs à travers un enseignement devant être particulièrement actif.

Le programme I.S.I. est donc ambitieux. L'approche pédagogique proposée est très différente de celle utilisée dans plusieurs autres disciplines d'enseignement. Elle est cependant réaliste en raison des facteurs suivants:

- La motivation initiale des élèves.
- La présence d'un ordinateur.
- Il s'agit de « faire », de « réaliser ».
- La relation professeur-élèves est changée: le professeur aide à réaliser.
- Chacun peut faire quelque chose de différent de son voisin.
- Les aspects théoriques se dégagent de la pratique et viennent faciliter l'action.

Il est intéressant de noter que les caractéristiques précédentes pourraient s'appliquer à des cours d'une discipline artistique. Il s'agit pourtant d'une discipline scientifique. Le paradoxe apparent provient du fait que l'ordinateur permet la réalisation « de projets personnels » à travers une méthode et un langage scientifique.

Les six facteurs précédents doivent donc subsister durant l'ensemble des cours. Comment? C'est le problème pédagogique.

2. Suggestion d'une approche

Ce qui précède montre qu'on ne peut enseigner la science de l'informatique comme on enseigne les mathématiques, par exemple; on ne peut présenter la « théorie » et demander ensuite de « l'appliquer ». On ne peut pas non plus privilégier une approche ressemblant à ce qu'on trouve souvent dans le secteur de l'enseignement professionnel: le contact avec la machine ne suffit pas. Il s'agit d'acquérir des concepts, et non pas seulement des savoir-faire. Il faut intégrer cependant ces deux tendances, un peu comme le font les professeurs de physique qui réalisent des expériences et qui donnent des cours théoriques. La comparaison ne peut cependant être poussée trop loin. L'ordinateur permet une expérimentation beaucoup plus rapide que cela n'est possible en physique. Le micro-ordinateur est un instrument interactif par nature, et ceci fait que la pédagogie que l'on doit concevoir pour son environnement soit réellement originale.

Face au comportement spontané des élèves devant la machine, il peut être tentant de combattre cette attitude par des cours de méthodologie de la programmation. Il y a peu

de chance que ceci soit efficace, le goût des élèves pour la machine étant beaucoup trop fort.

2.1 Le premier contact

On peut, par contre, offrir aux élèves un premier programme *interactif* portant sur un champ choisi par l'enseignant: dialogue, dessin, mise en page, chaîne de caractères. . . Le premier contact permet de se familiariser avec le clavier et les premières commandes de systèmes. Il permet de prendre conscience des possibilités de la machine, de l'écran, de l'utilisation des disques. Ce contact peut être l'occasion de présenter rapidement les diverses composantes du matériel et de leurs fonctions.

2.2 Choix d'un sous-ensemble d'instructions

Dans un deuxième temps, les élèves vont programmer et il faut éviter de les noyer sous un nombre trop grand d'instructions. On doit dès le début garder à l'esprit qu'il s'agit de fournir une méthode de travail et de demander aux élèves *d'acquérir une pensée structurée* et non pas une connaissance de détails. Il faut cependant qu'ils puissent réaliser quelque chose d'intéressant. On peut donc leur donner un sous-ensemble d'instructions (3 ou 4) touchant un domaine particulier. Ce peut être le même domaine que celui abordé par le premier programme interactif. À partir de ce mini-langage, il est possible de réaliser quelques programmes simples.

2.3 Les interventions du professeur visent à structurer le travail de l'élève

Dès le premier programme, on peut insister sur la documentation: titre, rôle, sens des variables. Les élèves en général acceptent bien de travailler la présentation de leur première oeuvre: c'est quelque chose qu'ils aiment montrer.

Sur le même thème, avec les mêmes instructions, d'autres idées de programmes peuvent surgir. Certains de ces programmes seront analogues, d'autres plus riches, englobant souvent les programmes déjà faits. Les diverses parties du programme seront séparées par des blancs.

Dans un premier temps, la programmation se fait directement au terminal: cette phase de « programmation sauvage » semble inévitable. Elle permet d'ailleurs d'expérimenter le fonctionnement de l'ordinateur et des instructions. Pendant ce premier stade, la documentation du programme se fait donc après que le programme fonctionne. De même, la description du fonctionnement du programme vient expliciter le déroulement d'un programme qui tourne déjà. C'est donc au professeur de le demander, ce qui permet d'introduire l'idée du pseudo-langage. Le professeur pourra demander dès le premier programme un listage du programme documenté et accompagné de sa description en pseudo-langage.

On se trouve à mettre en place des instruments qui serviront plus tard à analyser des problèmes et qui deviendront des outils de programmation.

2.4 Introduction des instructions de contrôle

Les premiers programmes sont purement séquentiels. Il faudra bientôt introduire les structures de répétition, les structures de sélection et les branchements inconditionnels.

Cela peut se faire lorsque les élèves en ont besoin pour réaliser le petit programme dont ils ont eu l'idée. Très vite ils auront besoin de ces structures et il faut profiter du fait qu'ils n'ont à manipuler que quelques instructions du langage pour aborder les

instructions de contrôle. L'introduction de ces instructions viendra donc simplifier un petit problème que l'élève veut résoudre.

On peut utiliser le pseudo-langage pour analyser le problème.

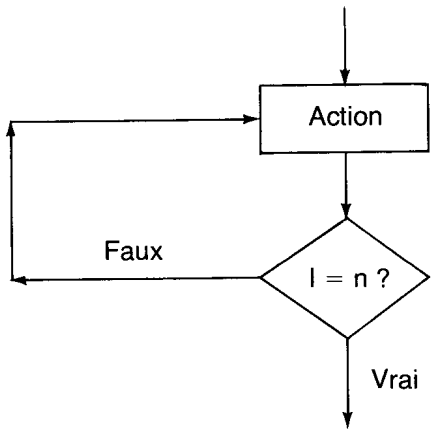
Exemple:

```
On veut:
Répéter
   Une action
Jusqu'à ce que. . .
```

Supposons que l'on veuille introduire un compteur pour réaliser cette répétition (il faut préciser ce que l'on va mettre à la place des pointillés).

```
I ← 0
Répéter
   Une action
   Augmenter I de 1
Jusqu'à ce que I = n
```

On peut alors représenter la structure « répète » par un organigramme (ou ordinogramme).



À propos de l'ordinogramme, il faut constater qu'il se prête bien à la décomposition d'actions élémentaires. Il n'est pas, par contre, un outil d'analyse d'un problème; c'est beaucoup plus un outil de documentation dont l'utilisation demeure appropriée pour de petits problèmes. Le pseudo-langage est, par contre, un outil d'analyse sérieux qui mérite toute l'attention souhaitée.

Il reste à traduire maintenant dans le langage utilisé.

On voit les étapes, et le professeur les utilise dans sa façon de procéder.

- La première analyse du problème se fait en langage courant.
- On la résume et la structure en pseudo-langage, lequel est très près du langage courant.
- On y ajoute des représentations graphiques si nécessaire.
- On traduit en utilisant le langage de programmation.

Le langage de programmation n'intervient qu'à la fin: c'est donc bien une initiation à une structure et non à un langage que l'on fait.

À la fin de cette première phase du travail, l'élève aura appris à manipuler l'ordinateur et aura été en contact avec les structures de contrôle. Il saura documenter un programme et *aura vu* son professeur utiliser le pseudo-langage pour structurer les programmes. Lui-même aura résumé le fonctionnement de ses programmes de cette façon. Il connaîtra de plus quelques instructions du langage et son comportement sera sans doute tout aussi sauvage; mais les éléments qui l'amèneront à surmonter les problèmes plus importants qu'il aura à affronter seront déjà en place.

Pendant cette phase le contact avec l'ordinateur et la technique « essai-erreur » sont largement utilisés. Ceux-ci permettent de se mettre dans le contexte d'un apprentissage naturel, un peu comme celui d'une langue maternelle et aussi de surmonter « la peur du terminal ».

De temps à autre, le professeur peut faire le point sur ce qui a été fait, fournir quelques documents d'appoint sur les structures de contrôle, le pseudo-langage, donner quelques exemples d'organigrammes et de plan de programmes. Tout cela peut constituer un matériel auquel l'élève peut se référer. Cependant, il s'agit de minimiser le temps consacré à de telles activités; l'essentiel du travail se réalise au terminal, au début tout au moins.

L'acquisition d'une méthode de travail, de structures de pensée, ne peut être que le résultat d'une pratique orientée par le professeur. Rien n'est encore acquis. Il faudra utiliser les techniques qui viennent seulement d'être abordées. Chaque nouveau programme sera l'occasion de les utiliser et donc de les acquérir progressivement.

2.5 Traitement des erreurs

Lorsqu'une erreur survient, l'ultime recours est le professeur. Or, la capacité de trouver et de corriger ses erreurs fait partie intégrante de l'apprentissage de la programmation. C'est aussi une des composantes fondamentales de *l'autonomie intellectuelle* que l'on veut développer chez les élèves. Plutôt que de corriger les erreurs des élèves, il faut leur apprendre à les trouver. Selon l'erreur, on doit apprendre à vérifier la syntaxe de l'instruction, ou à simuler l'exécution du programme en dehors du terminal. Souvent, les élèves ont du mal à accepter que l'ordinateur soit une machine qui ne fait aucune interprétation, qui fait strictement ce qu'on lui dit de faire. Pour détecter les erreurs de structure, l'utilisation du pseudo-langage est souvent très efficace. Dès le début, il est bon aussi d'amener les élèves à se poser la question: « qu'est-ce que ce programme *change à l'état des variables?* » plutôt que la question « qu'est-ce que mon programme fait? ». Ces attitudes ne sont pas spontanées et c'est le professeur, par ses questions, qui peut les induire chez les élèves. C'est aussi un avantage que les messages d'erreurs apparaissent en français, même si le langage utilisé fait appel à un vocabulaire pseudo-anglais.

2.6 Les risques

Pour pouvoir réaliser des projets intéressants, il faut avoir à sa disposition des moyens assez puissants et le professeur qui accepterait de se lancer avec des élèves dans un projet sans une préparation adéquate risque une expérience décevante.

- Un projet trop compliqué ne sera jamais réalisé.
- Sa réalisation sera beaucoup plus pauvre que prévue.
- Des difficultés au niveau des détails de la programmation empêcheront de s'attacher à la structure du programme.
- Il est difficile de suivre un grand nombre de projets différents, surtout lorsqu'ils sont mal documentés et faibles dans leur structure.

Il peut en résulter un désintérêt progressif et les objectifs du programme ne seront pas atteints. Le professeur doit donc se préparer afin d'éviter tous ces écueils.

2.7 Restreindre le domaine des projets, non leur ampleur

Dans un premier temps, il est possible de se restreindre à un certain nombre de sujets. Au début, le nombre des instructions avait été limité. Il est possible maintenant de se limiter à un domaine d'activités tel que: les jeux aléatoires, les applications à la physique, le graphisme, l'animation, les chaînes de caractères, la musique, etc. S'il paraît important de restreindre le domaine des projets pour des raisons pratiques, il faut par contre que les élèves puissent aller le plus loin possible dans chacun des domaines choisis.

2.8 Écrire des procédures ou des sous-programmes permettant des réalisations intéressantes

Le domaine étant limité, le choix des possibilités de projets reste encore grand. D'autre part, il est important que les élèves puissent obtenir des résultats intéressants assez rapidement et ne se laissent pas submerger par des technicalités mineures qui vont les empêcher d'acquérir des notions plus importantes. On peut leur fournir des instruments facilitant le travail soit sous forme de sous-programmes, ou de procédures qui fourniront des outils plus puissants que le langage de base de leur ordinateur.

Exemple: des sous-programmes simulant des lancers de dés, permettant de générer certains caractères, appelant les notes de musique que l'on désire, etc.

L'ensemble de sous-programmes mis à leur disposition est fonction, naturellement, de l'initiative du professeur. Il dépend de la machine utilisée, du langage, des périphériques disponibles. Les sous-programmes doivent considérablement alléger l'écriture des programmes, donc le suivi du professeur et le travail des élèves. Ils ont le mérite, également, d'initier à une forme modulaire de programmation.

2.9 Fournir une documentation écrite

On peut aussi ajouter des exemples de projets: il ne faut cependant pas aller trop loin dans ce domaine pour ne pas stériliser les idées. Ceci doit sans doute être fait à la demande, dans le cas où un élève serait sans idée, ce qui est rare.

Il est bon aussi que chaque élève ait à sa disposition un document dans lequel il pourra trouver les commandes du système le concernant, les instructions, un rappel des structures de contrôle, des exemples d'utilisation de pseudo-langage, des programmes documentés.

2.10 Participer à la définition du projet

Un élève a souvent une idée de projet, plutôt qu'un projet. Le travail du professeur est donc de préciser le projet, en général de le restreindre, de le réorienter selon les possibilités du matériel, du langage et des sous-programmes fournis. Cela peut se faire oralement et relativement rapidement. Il faut que l'élève prenne rapidement conscience de l'ampleur de son futur travail.

On peut lui demander, par écrit, sous forme de schéma ou de quelques phrases, un plan de déroulement de son travail. Le plan peut être très global. Il est cependant important car il concrétise une certaine façon de vouloir résoudre le problème, et c'est précisément un des objectifs explicites du cours.

2.11 La réalisation du projet: Amener à structurer

Lors de la réalisation d'un premier projet, l'envie de programmer est encore très forte. Le mieux est donc de choisir une subdivision du projet global et d'essayer de la réaliser sur la machine. Le travail du professeur consiste à s'assurer que pour cette partie du programme, les élèves vont utiliser les techniques de programmation décrites plus haut. L'important est que le produit fini soit structuré, qu'un plan soit facile à dégager et apparaisse dans l'écriture même du programme.

2.12 La notion de procédure ou de sous-programme

C'est la partie importante. Il faut qu'elle soit introduite dès le premier projet, dans des cas simples où le problème de la transmission des paramètres ne se pose pas. Le professeur peut lui-même transformer une partie de programme.

3. D'autres types de projet

3.1 Le projet coopératif

On peut vouloir insister sur le fait qu'un projet est un ensemble de sous-projets reliés entre eux en réalisant effectivement cette division des tâches à l'intérieur de la classe. C'est le « projet coopératif ». Pour cela on choisit un projet unique, d'intérêt, pour toute la classe. L'analyse du projet se fait en commun et il en résulte un plan d'action. La classe est alors divisée et chaque sous-groupe a une partie du projet à réaliser. Le travail de chaque sous-groupe se matérialise sous la forme d'une unité de programme autonome. Le programme final se fait encore en commun: il s'agit d'organiser les diverses unités du programme. On peut ainsi aborder pour tous les élèves ensemble les problèmes survenant lors de la réalisation de programmes structurés en sous-programmes (ou en procédures) en renumérotant les lignes pour les placer après le programme principal. On obtient ainsi des programmes plus courts. On peut aussi énoncer la règle « pas d'unités de programme dépassant les dimensions de l'écran ».

À ce propos, rappelons que le rôle essentiel d'un sous-programme est de permettre une programmation claire, facile à lire et à corriger.

Les avantages d'une telle approche sont évidents: on sauve le groupe-classe, on peut aborder les mêmes difficultés en même temps, la communication entre les élè-

ves reste bonne, et l'on simule dans la classe le fonctionnement d'un programme structuré.

3.2 Les projets autour d'un thème

On choisit un thème correspondant à un domaine de l'activité humaine; réalisation d'un journal grâce à l'ordinateur, le traitement de texte, l'ordinateur et la banque. . . La classe entière se documente sur ce sujet. Il est possible d'organiser des visites, de faire venir de la documentation. À partir de là, on peut structurer la classe en vue de la réalisation d'un projet coopératif, dans lequel on peut demander aux élèves d'imaginer leurs projets en fonction de ce qu'ils ont vu. On se trouve donc à explorer un domaine où l'ordinateur joue un rôle important et en plus de la programmation, on apprend aussi à quoi sert l'ordinateur dans le monde du travail.

3.3 Le projet interdisciplinaire

On prend cette fois-ci une autre matière: le français, la physique, l'anglais et l'on identifie des projets pouvant être utilisés ensuite dans ces matières. Le travail, cette fois, peut s'amorcer avec des professeurs des autres disciplines. Les programmes pourront être effectivement utilisés. Ceci peut être réalisé partiellement en mettant directement en relation des élèves plus avancés avec les autres professeurs. On peut ainsi régler le problème des élèves ayant un ordinateur à la maison.

On peut aussi concevoir des projets utilisant des connaissances dans plusieurs disciplines: un peu de physique, quelques théorèmes de mathématiques, la connaissance de certaines règles du français peuvent être utiles pour réaliser des « tirs au canon », des « simulations de satellites », des « jeux grammaticaux ». Des projets de ce type permettent aussi d'explorer des connaissances qui seront vues plus tard, ou qui ont été déjà abordées.

3.4 Les projets qui n'entrent dans aucune catégorie

C'est le type de projet non prévu, suggéré par un élève. Il faut savoir si le professeur se sent la capacité de suivre un tel projet, bien qu'il soit « hors cadre ». Certains élèves plus expérimentés peuvent avoir plus d'ambition que les autres, en particulier s'ils disposent déjà d'un ordinateur. Parmi ceux-là, certains sont particulièrement brillants, et ils n'ont finalement besoin de personne. C'est une minorité. Les autres connaissent surtout un tas de « trucs » reliés à la marche de la machine. Ils programment en général de façon illisible, mais veulent continuer à éblouir leurs camarades par une grande virtuosité. Il vaut mieux les diriger dans un domaine qu'ils ne connaissent pas afin de leur apprendre à structurer leur travail. On peut aussi les utiliser quelque temps comme opérateur auprès des autres.

3.5 Le projet boule-de-neige

L'essentiel est que chaque élève puisse réaliser un projet qui lui convienne. C'est la réussite qui est la meilleure motivation. L'ordinateur peut être l'occasion pour certains de se révéler à leurs yeux et aux yeux des autres. Il faut donc aborder des domaines variés, permettre la réalisation de projets simples, qui sont cependant des projets et qui sont définis en grande partie par l'élève; c'est lui qui analyse, qui programme et qui montre ce qui devient une oeuvre personnelle. Au début surtout, il faut pouvoir éprouver la joie de la réalisation d'un logiciel qui fonctionne. Il faut pouvoir éprouver cette joie rapidement, et ceci constitue une caractéristique de l'approche préconisée dans ce programme. Les premiers projets doivent donc avoir des objectifs limités, partiellement réalisables. Il y a des succès partiels qui seront la raison qui poussera à une réussite plus grande.

Lors de la première partie du programme surtout, un premier petit projet réussi donnera l'envie de l'enrichir une première fois, puis une seconde, jusqu'à ce qu'on obtienne un projet réalisé complexe. Dans ce cas, il n'y a pas d'analyse descendante. On programme au contraire de façon ascendante. Il est possible de le faire de façon structurée, et c'est ce qui est important. Cette approche, plus pragmatique, peut donner le plaisir de la réussite. Le petit programme diminue les difficultés liées à une analyse d'un problème complexe. Réutiliser un programme déjà fait dans un programme plus complexe est une démarche très structurante. Elle correspond de plus à un mouvement naturel de la pensée et elle permet de justifier, par la suite, la nécessité d'une analyse descendante lorsqu'un problème complexe est abordé.

Le rôle du professeur est de veiller qu'à chaque stade, le programme forme un tout, qu'il soit documenté et décrit en pseudo-langage. Il faut que les ajouts successifs se fassent sous forme d'unités autonomes et non pas en modifiant de façon désordonnée des programmes qui fonctionnaient déjà, qui vont donc devenir illisibles, et en général, ne plus tourner. Il faudra sans doute à un certain moment décider de repartir à zéro: ce sera fait à partir d'un plan, d'autant plus facile à écrire que les unités auront été écrites auparavant.

4. Les objectifs sociaux et affectifs

La plupart des programmes mentionnent de tels objectifs, bien que les moyens de les atteindre ne soient pas toujours très explicites. Ce n'est pas le cas de celui-ci qui offre des possibilités concrètes de faire évoluer l'idée que les élèves ont d'eux-mêmes et d'améliorer la communication entre les élèves d'une part, entre les élèves et le professeur d'autre part.

4.1 L'image que l'élève a de lui-même

Quel que soit le type de projet choisi, il doit exister des moments où l'élève se trouve seul au terminal pour écrire ou tester un programme. Sans être jugé par personne d'autre, son tête-à-tête avec la machine lui permet de faire son chemin personnel vers la réussite. L'ordinateur ne juge pas l'utilisateur. Une erreur n'est que le signe qu'il y a du travail à faire pour que le programme tourne. Ce n'est pas la phase ultime, celle qui est jugée. Le professeur doit s'attacher à préserver cet aspect du dialogue élève-ordinateur. La confiance en soi peut s'en trouver augmentée, surtout si les interventions de l'enseignant visent à permettre que l'élève surmonte des difficultés qui finiraient par le décourager.

Cette confiance en soi se trouve augmentée au fur et à mesure que la connaissance de l'ordinateur s'approfondit: connaissance technique, mais aussi du rôle que l'ordinateur peut jouer. Lecture d'articles, de livres, travaux de recherche, films aident à le comprendre également.

4.2 La communication avec les autres

Certains élèves peuvent avoir tendance à s'enfermer dans leur projet. Au bout d'un certain temps, ni le professeur, ni leurs camarades savent ce qu'ils font. Si en plus, la programmation qu'ils réalisent est difficile à lire, ils vont être livrés à eux-mêmes et les objectifs du cours ne seront pas atteints. Le professeur doit donc se voir comme un gestionnaire de projets. Il doit donc être capable de suivre l'évolution de chaque projet. Pour cela, il peut construire une grille qui lui permettra de suivre l'évolution de la planification et de la réalisation des projets. Les élèves eux-mêmes doivent pouvoir prendre conscience de l'évolution de leur projet par rapport aux prévisions et dégager les raisons qui justifient les écarts éventuels. C'est en les amenant à décrire ce qu'ils font qu'on peut les amener à réfléchir et à améliorer leurs habitudes. La tenue d'un journal de bord très succinct, mais lisible, peut être d'un grand secours. On pourra y

trouver les intentions, les plans successifs, les remarques concernant les problèmes rencontrés, les changements d'orientation, les résumés des programmes en pseudo-langage. De temps à autre, toute la classe peut échanger sur les problèmes rencontrés et ce peut être l'occasion de mettre en évidence des techniques plus efficaces, des moyens différents de résoudre des problèmes, ou des structures de programmation. Certains seront sûrement très heureux de parler de ce qu'ils ont fait aux autres. Ce peut être une façon de débuter chaque séance de travail: chaque groupe fait le point et indique comment il voit l'évolution de son travail. En fin de session, on peut prévoir une journée où les élèves montreront aux autres classes leur travail. C'est à la fois une façon de faire de la publicité pour ces cours, et aussi de favoriser la présentation la plus claire possible de chaque projet. Le projet communautaire est sans doute celui qui favorise le plus l'échange puisque c'est toute la classe avec l'aide du professeur qui analyse et planifie toutes les tâches à réaliser, lesquelles seront menées à bien par de petites équipes qui devront ensuite regrouper leurs travaux pour réaliser le projet initial.

La connaissance de l'histoire de l'ordinateur, de sa place dans la société d'aujourd'hui et de demain, le développement de l'esprit critique face à ses utilisations peuvent être prétextes à discussions, à des travaux de recherche qui doivent améliorer aussi la communication entre les élèves eux-mêmes.

Enfin, la relation entre le professeur et ses élèves peut être bien différente de ce qu'elle est dans bien des cours. Ici, le professeur se trouve à aider les élèves à réaliser un projet qui leur plaît. C'est par le dialogue, en particulier, que peuvent passer les idées de résolutions de problèmes. Ce peut être un simple conseil qui va conduire à la réussite; et parce qu'il y a eu réussite, l'élève s'en souviendra.

5. Résumé

La dynamique du cours provient du désir des élèves de réaliser des projets plus ou moins importants, plus ou moins communautaires. C'est à travers l'action que sont atteints les objectifs, grâce à l'intervention du professeur qui aide à la structuration de la démarche des élèves, de la définition du projet à la fabrication du logiciel.

DÉMARCHE DE L'ÉLÈVE

INSTRUMENTS CONCEPTUELS ET MÉTHODOLOGIQUES

OBJECTIFS ATTEINTS

IDÉE DE PROJET

TECHNIQUE DE RÉSOLUTION DE PROBLÈMES

CONSTRUIRE DES ALGORITHMES

STRUCTURATION DU PROJET

PSEUDO-LANGAGE

EXERCER DES HABILETÉS DE RÉSOLUTION DE PROBLÈMES

STRUCTURATION DE CHAQUE PARTIE

PROCÉDURES

SOUS-PROGRAMMES

VARIABLES GLOBALES

CHAQUE PARTIE EST À NOUVEAU STRUCTURÉE

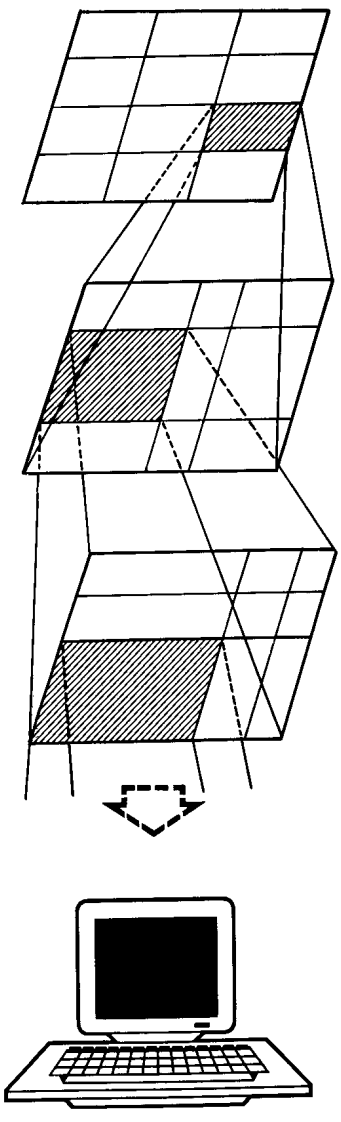
VARIABLES LOCALES

PROGRAMMER L'ORDINATEUR

STRUCTURES DE RÉPÉTITION

ETC.

PRODUCTION DU LOGICIEL



III - L'évaluation

Le problème de l'évaluation est abordé de façon globale dans le programme. Il est donc inutile de répéter ce qui s'y trouve.

Pour le professeur dans sa classe, l'évaluation a deux fonctions essentielles.

- Poser un diagnostic final sur ce que l'élève sait faire à un moment donné.
- Aider l'élève dans sa démarche en lui faisant prendre conscience de ce qu'on attend de lui, où il se situe par rapport à cette attente et quels sont les moyens à sa disposition pour réussir à progresser.

Il s'agit donc d'être prudent dans le choix des moyens d'évaluation. Il arrive en effet très souvent que l'évaluation décide dans les faits de l'orientation de tout un cours. Les élèves en effet veulent réussir l'examen ou le test et ils orienteront leurs efforts dans ce sens. Si l'examen ne porte que sur une partie des objectifs du cours, les autres seront négligés. Ceci est particulièrement important dans ce programme qui veut développer une façon de travailler bien plus que des connaissances ponctuelles.

Dans la mesure où l'on veut développer l'autonomie intellectuelle, il faut pouvoir donner des moyens d'auto-évaluation aux élèves qui vont leur permettre d'exercer leur jugement. Ce pourrait être une grille du genre de celle que l'on trouve dans le document « Quelques lignes directrices concernant l'évaluation de projets d'élèves ».(1) Dans tous les cas, cette grille devrait faire référence à une démarche globale partant de la définition du projet et allant jusqu'à la fabrication du logiciel. L'élève devrait pouvoir se situer par rapport à cette grille et expliquer pourquoi sa démarche s'écarte de la démarche suggérée. Dans bien des cas cet écart se justifie, mais l'important est de pouvoir en donner les raisons.

Si le professeur désire donner des notes, il doit prendre en compte l'ensemble de la démarche. Sa note devra donc tenir compte du résumé du projet, exprimé en langage courant, qui doit être assez clair et précis pour être compréhensible et réalisable; du plan du projet à réaliser, dans lequel devront être identifié un ensemble de tâches à réaliser, et de leurs liens; de l'échéancier, et de la concordance de la réalité et de l'échéancier; de la formulation des tâches en pseudo-langage, de la documentation du programme, de sa lisibilité, de l'originalité du projet, etc. Sa note devra aussi tenir compte des progrès de l'élève, mais ne pas briser son enthousiasme. Peut-être, mieux vaut s'en tenir à l'auto-évaluation au début, ce qui donne le temps à l'élève de prendre conscience de la démarche qu'on attend de lui, et d'introduire les notes chiffrées un peu plus tard. Ceci se justifie d'autant plus que l'ordinateur donne déjà une évaluation immédiate. Le programme marche ou ne marche pas. Il faut laisser le temps à l'élève de prendre confiance en lui avant d'intervenir avec des notes. L'ordinateur ne juge pas, pourtant il est impitoyable et n'admet pas l'à-peu-près.

La note finale ne peut pas provenir d'un examen final, mais doit prendre en compte le travail réalisé sur les projets de l'année. L'examen portera donc sur les objectifs 3, 4 et 5 et à ce propos rappelons que le programme propose que cet examen corresponde à 30% de la note finale.

(1) Voir la bibliographie

Bibliographie

LABELLE, Marcel, TAURISSON, Alain, *Quelques lignes directrices concernant l'évaluation de projets d'élèves,* (Document interne, PERMAMA, Université du Québec, 1979).

LAURENT, J.D., *Initiation à l'analyse et à la programmation,* Dunod (1982).

LEDGARD, H.F., (Traduit et annoté par J. Arsac), *Proverbes de programmation,* Dunod (1978).

NEVISON, John M., L'art de bien programmer en basic « *Le petit livre du style* », Eyrolles (1981).

PAPERT, S., *Jaillissement de l'esprit,* (Traduction de Mindstorms: Children Computers and Powerful Ideas), Paris: Flammarion (1981).

WIRTH, N., *Introduction à la programmation systématique,* Masson (1980).

8384-978

Gouvernement du Québec
Ministère
de l'Éducation