UNIVERSITE DE LIEGE

Travaux publiés par le
LABORATOIRE D'ANALYSE STATISTIQUE
DES LANGUES ANCIENNES

Sous la direction du Professeur Louis DELATTE

LE TRAITEMENT AUTOMATIQUE DE L'INFORMATION EN SCIENCES HUMAINES

INITIATION AU LANGAGE PL/1

J. DENOOZ
Maître de Conférences

LIEGE
1979

INTRODUCTION

LES ASPECTS TECHNIQUES DU TRAITEMENT DE L'INFORMATION

Le mot "information" désigne ce que l'homme peut *percevoir* par l'intermédiaire de ses organes sensoriels et *identifier* à l'aide d'un langage. En outre, l'être humain capable de percevoir et d'identifier l'information peut également *communiquer* des informations. La *perception*, l'*identification* et la *communication* sont les trois aspects fondamentaux du traitement de l'information tant au niveau de l'homme qu'au niveau de l'ordinateur.

1.- **La perception** : pour percevoir des informations, l'ordinateur est doté d'organes que l'on appelle des unités d'entrée ou unités de saisie. Ces unités transmettent les données à partir d'un support matériel vers l'unité centrale de traitement de l'ordinateur. Les principales unités d'entrée sont :

a) le lecteur de cartes mécanographiques;
b) le lecteur de rubans perforés;
c) le dérouleur de bandes magnétiques;
d) l'armoire à disques magnétiques;
e) l'unité de lecture optique;
f) l'écran cathodique;
g) le disque souple.

A chacune de ces unités correspond un support différent de l'information; cependant, quel que soit le type de support, l'information est toujours représentée, dans l'unité centrale, par une série d'impulsions électriques. Voici la description des principaux supports de l'information.

a) La carte mécanographique est un carton isolant de 187,3 mm de long et de 82,5 mm de large. Elle se divise verticalement en 80 colonnes numérotées de 1 à 80 et horizontalement en 12 niveaux ou lignes. Chaque colonne peut recevoir la marque d'un caractère d'imprimerie numérique ou alphabétique sous la forme d'une ou plusieurs perforations. La signification d'une perforation dépend du niveau où elle se situe. La position des colonnes et des niveaux est indiquée sur les cartes de modèle standard, par des chiffres imprimés, sauf pour les deux niveaux situés dans la partie supérieure de la carte.

Une perforation à l'un des trois premiers niveaux (12, 11 et 0) est appelée perforation hors-texte, une perforation à l'un des niveaux de 0 (zéro) à 9 est appelée perforation numérique. On remarquera le caractère mixte de la perforation 0 qui joue le rôle de hors-texte pour les signes alphabétiques et le rôle de perforation numérique pour les chiffres.

Les chiffres sont représentés par une des dix perforations numériques. Quant aux caractères alphabétiques, ils exigent la présence dans une même colonne d'une perforation hors-texte (12, 11 ou 0) et d'une perforation numérique (1 à 9).

L'alphabet a été réparti en trois groupes selon la perforation hors-texte utilisée :

A - I hors-texte 12
J - R hors-texte 11
S - Z hors-texte 0

A l'intérieur d'une même zone, chaque lettre est distinguée par une perforation numérique. Voici le détail du code alphabétique.

A 12 et 1	J 11 et 1	
B 12 et 2	K 11 et 2	S 0 et 2
C 12 et 3	L 11 et 3	T 0 et 3
D 12 et 4	M 11 et 4	U 0 et 4
E 12 et 5	N 11 et 5	V 0 et 5
F 12 et 6	O 11 et 6	W 0 et 6
G 12 et 7	P 11 et 7	X 0 et 7
H 12 et 8	Q 11 et 8	Y 0 et 8
I 12 et 9	R 11 et 9	Z 0 et 9

La perforatrice ou poinçonneuse est une machine à clavier qui permet de perforer les informations dans les cartes.

b) Le ruban perforé est une bande de papier fort dont les dimensions varient en fonction du type de matériel. La longueur d'un ruban peut atteindre de 250 à 300 mètres tandis que sa largeur se situe généralement entre 4 et 5 centimètres. Ce support connaît actuellement un certain succès en imprimerie car il permet un passage aisé à la photocomposition.

c) La bande magnétique est constituée d'un film souple. Elle se présente sous la forme d'un ruban qui a, en général, environ 730 mètres de long (2400 pieds) et 1,27 centimètre de large (un demi-pouce). La quantité d'informations que l'on peut enregistrer sur une bande magnétique dépend de sa densité, c'est-à-dire, du nombre de caractères que l'on peut stocker sur un *inch*. Actuellement, la densité la plus répandue est de 1600 caractères par inch (1600 bytes per inch ou encore 1600 bpi). Une bande magnétique comprend, dans sa largeur, 7 ou 9 canaux qui peuvent contenir chacun une information binaire.

Les informations peuvent être enregistrées sur une bande, soit directement à l'aide d'une encodeuse, soit par l'intermédiaire d'un fichier de cartes que l'ordinateur lit pour le retranscrire sur la bande.

d) Les disques magnétiques se composent de plusieurs plateaux circulaires superposés qui tournent sur un axe. La face supérieure et la face inférieure des premier et dernier plateaux sont inutilisées pour des raisons de sécurité. Chaque face est divisée en un certain nombre de parties qui constituent des pistes. Les pistes d'un même plan vertical forment ensemble un cylindre.

L'armoire à disques possède un bras mobile muni de têtes magnétiques qui chacune peuvent lire ou écrire les informations sur une des faces.

e) Le support de la lecture optique est un papier standardisé sur lequel sont inscrits des caractères. Actuellement l'emploi de la lecture optique reste limité, relativement peu souple et extrêmement coûteux.

f) L'écran cathodique est constitué d'un clavier semblable à celui d'une machine à écrire et d'un écran de visualisation comparable à un écran de télévision. Il permet le dialogue direct avec l'ordinateur. Cette unité rend possible la vérification et la

correction immédiate des informations dactylographiées sur le clavier puisque toute donnée apparaît sur l'écran.

g) Le disque souple est un support magnétique sur lequel on peut stocker une quantité d'informations équivalente à 2.000 cartes perforées.

Pour terminer ce paragraphe relatif aux supports de l'information, il me semble utile de dire un mot de leur évolution.

A l'origine des supports de l'information, on trouve les systèmes à cartes perforées dont l'utilisation remonte à la dernière décennie du siècle passé et est due au Dr. *HOLLERITH*. Les principaux avantages de la carte sont la facilité de correction, une bonne conservation et un prix peu élevé.

Le deuxième support dans l'ordre chronologique est le ruban perforé qui, par rapport à la carte, présente l'avantage de pouvoir contenir, sous une forme réduite, un très grand nombre d'informations.

Dans la suite, on trouve les unités magnétiques de stockage de l'information, les bandes et les disques. Ces supports répondent à une nécessité évidente : la quantité d'informations à traiter et à conserver croît sans cesse et exige des supports à forte capacité.

Enfin, le développement des supports magnétiques et l'utilisation du tube cathodique pour visualiser l'information ont permis la saisie directe de données sur disques magnétiques à partir des terminaux à écran et à clavier.

2.- L'identification de l'information se fait au niveau de l'unité centrale de traitement. Chaque caractère est représenté sous une forme qui repose sur le calcul binaire. Les aspects techniques de l'unité centrale de traitement exigeraient un long exposé. Je me bornerai à quelques généralités sur la structure interne de l'ordinateur IBM 370.

La cellule de base de cette machine est l'*octet*. C'est la position de mémoire capable de contenir *un caractère*. Cette position se divise en 9 parties ou *bits*. Le premier *bit* sert à contrôler la parité de l'octet, les huit autres contiennent l'information. Ces *bits* sont positionnés à 1 ou 0 suivant que le courant y passe ou n'y passe pas. La combinaison des *bits* 1 et des *bits* 0 dans un même octet donne à ce dernier sa valeur. En voici quelques exemples :

0000 0001	valeur 1	0000 0110	valeur 6
0000 0010	valeur 2	0000 0111	valeur 7
0000 0011	valeur 3	0000 1000	valeur 8
0000 0100	valeur 4	0000 1001	valeur 9
0000 0101	valeur 5		

Chaque octet est numéroté et cette numérotation permet de retrouver un octet déterminé et, par conséquent, son contenu. De même, le numéro de l'octet peut être utilisé pour faire exécuter à l'ordinateur la série d'instructions qui commence à une position précise de l'unité centrale.

3.- **La communication** des résultats d'une tâche effectuée par l'ordinateur se fait par l'intermédiaire des unités de sortie. Parmi ces unités, je signalerai les suivantes.
a) le perforateur de cartes;
b) l'imprimante qui fournit les résultats sous une forme écrite. Les imprimantes les plus répandues travaillent à environ 1.000 lignes à la minute. Cependant des imprimantes à *laser* apparaissent progressivement; elles atteignent des performances supérieures à 12.000 lignes par minute;
c) les disques et les bandes magnétiques sur lesquels on stocke les résultats à réutiliser ultérieurement. Les disques et les bandes que l'on utilise, soit comme unités d'entrée, soit comme unités de sortie, sont appelés mémoires auxiliaires ou plus souvent *mémoires périphériques*;
d) l'écran cathodique.

Unités d'entrée, unité centrale de traitement et unités de sortie forment ensemble l'ordinateur, la machine à traiter l'information. Cet ensemble, quelque perfectionné qu'il soit, ne peut exécuter un travail que s'il reçoit au préalable des ordres précis.

LES SYSTEMES LOGIQUES

A bien des égards, l'ordinateur ressemble à un jeune enfant. Pour mener une tâche à bien, celui-ci a besoin de renseignements très détaillés : précisions sur les objets qu'il doit manipuler, localisation de ces objets, etc. Au surplus, le langage même que l'on utilise pour parler aux jeunes enfants doit être adapté à leur connaissance des choses.

Ainsi, l'ordinateur pour travailler doit également recevoir toutes les précisions sur les données à traiter et tous les ordres qui lui permettront d'effectuer la tâche qu'on lui demande. D'autre part, le langage employé pour communiquer avec la machine devra répondre à certaines exigences.

Un langage de programmation est un système symbolique destiné à décrire de façon logique et formelle les étapes du traitement automatique des informations. Un tel langage, purement conventionnel, rend possible la communication entre l'homme et la machine.

Comme les langages humains sous leur forme écrite, les langages de programmation se composent de mots et de signes qui constituent lorsqu'ils sont présentés dans un ordre déterminé et selon une syntaxe précise, des phrases et un texte compréhensible pour l'ordinateur. Les phrases, en langages de programmation s'appellent des instructions et un ensemble de phrases s'appelle un programme.

Un langage de programmation comprend deux types d'instructions :
a) *les instructions non-exécutables* : ce sont les déclarations. Elles servent à définir les zones de travail qui doivent figurer dans l'unité centrale pour enregistrer et traiter les données ainsi que les résultats des opérations. D'autre part, les déclarations servent à préciser les caractéristiques des fichiers (c'est-à-dire les collections de données) d'entrée et de sortie.

b) *les instructions exécutables* : ce sont les opérations grâce auxquelles l'ordinateur effectuera une tâche déterminée. On distingue cinq classes d'opérations.
1) les opérations d'entrée/sortie ou si l'on veut les opérations de lecture/écriture destinées respectivement à saisir et à communiquer les informations.
2) les opérations arithmétiques; ce sont les quatre opérations fondamentales.
3) les opérations de saut; ce sont des ordres qui permettent de passer d'un point du programme à un autre sans nécessairement suivre la séquence normale des instructions. Il y a deux types de saut, les sauts conditionnels et les sauts inconditionnels. Les premiers résultent d'une comparaison et sont destinés à effectuer un choix entre deux séries d'instructions. La comparaison a pour but de contrôler la relation existant entre deux termes. Soit les termes *A* et *B*, il y a entre eux huit relations possibles :

1	A =	B	A est égal à B
2	A ¬=	B	A est différent de B

3	A	>	B	A est plus grand que B
4	A	<	B	A est plus petit que B
5	A	¬>	B	A n'est pas plus grand que B
6	A	¬<	B	A n'est pas plus petit que B
7	A	>=	B	A est plus grand ou égal à B
8	A	<=	B	A est plus petit ou égal à B

Les seconds commandent le passage à une instruction du programme dans tous les cas.
4) les instructions d'affectation transportent dans une zone de la mémoire la totalité ou une partie d'une autre zone. Ainsi, l'opération $X = Y$ a pour effet de faire passer dans X le contenu de Y. Mais il faut bien noter que cette opération reproduit plutôt qu'elle ne transporte le contenu d'une zone puisque à l'issue de l'ordre $X = Y$, le contenu initial de Y se trouve simultanément dans les deux zones.
5) les instructions de manipulation des chaînes de caractères. Ces opérations, particulièrement développées en PL/I, langage de programmation dont je parlerai bientôt, sont destinées au traitement des données alphabétiques, donc au traitement des textes.

Pour terminer cette description générale des systèmes logiques, il me reste à dire quelques mots de la façon dont travaille un ordinateur. Le déroulement du programme dans l'unité de traitement est à la fois *séquentiel* et *répétitif*.

Le déroulement du programme est séquentiel en ce sens que les opérations sont exécutées dans l'ordre où elles se trouvent dans le programme.

Il est répétitif puisque le programme doit normalement être parcouru dans sa totalité pour une donnée déterminée et qu'il devra être répété autant de fois qu'il y a de données.

Lire et écrire des informations, les comparer, les transporter d'une zone à une autre et les compter, ce sont les seules opérations que l'ordinateur peut réaliser. C'est avec ces quelques ordres élémentaires que l'informaticien doit réaliser les tâches les plus complexes. En fait, c'est par une analyse approfondie et rigoureuse du problème que l'on fera reproduire le raisonnement humain par l'ordinateur.

L'ANALYSE DU PROBLEME

La seule manière d'effectuer une analyse correcte d'un problème est de suivre les principes énoncés par *DESCARTES* au livre II du *Discours de la Méthode*; on me permettra de les rappeler car c'est sur eux que repose entièrement la logique du traitement de l'information.

1) Le premier était de ne recevoir jamais aucune chose pour vraie que je ne la connusse évidemment être telle; c'est-à-dire d'éviter soigneusement la précipitation et la prévention, et de ne comprendre rien de plus en mes jugements que ce qui se présenterait si clairement et si distinctement à mon esprit que je n'eusse aucune occasion de le mettre en doute".

2) "Le second, de diviser chacune des difficultés que j'examinerais en autant de parcelles qu'il se pourrait et qu'il serait requis pour les mieux résoudre".

3) "Le troisième, de conduire par ordre mes pensées, en commençant par les objets les plus simples et les plus aisés à connaître, pour monter peu à peu comme par degrés jusqu'à la connaissance des plus composés".

4) "Et le dernier, de faire partout des dénombrements si entiers et des revues si générales, que je fusse assuré de ne rien omettre".

Pour s'aider dans la description d'un problème et de sa solution, l'informaticien utilise la technique des ordinogrammes. Il s'agit d'un procédé qui vise à représenter de façon claire et complète le raisonnement sous une forme graphique. Cette méthode se situe à mi-chemin entre le raisonnement humain qui est intuitif et la logique rigide de l'ordinateur.

En fait, l'ordinogramme est le programme dessiné. Il doit atteindre les buts suivants :

1) exprimer et permettre de suivre la ligne du raisonnement logique;
2) établir un lien entre la définition explicite du problème et le programme des instructions;
3) conduire à une transposition facile du raisonnement dans un langage compréhensible par la machine.

Un ordinogramme doit présenter les qualités suivantes qui rappellent à bien des égards les quatre principes de *DESCARTES* :

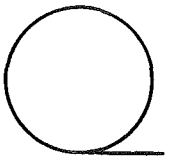
1) il doit être complet; il doit envisager tous les aspects du problème, c'est-à-dire :
   a) les particularités des données - leur type (numérique ou alphabétique);
      - leur place sur le support de l'information;
      - leur longueur.
   b) les unités d'entrée/sortie utilisées et les méthodes d'accès aux données.
   c) les étapes successives du traitement.
2) il doit être simple. La division en "parcelles" doit conduire à un ensemble d'éléments premiers qui peuvent apparaître dans certains cas comme des évidences.
3) il doit être clair.

L'établissement d'un ordinogramme se fait grâce à un certain nombre de figures qui symbolisent une des opérations que l'ordinateur peut exécuter. Voici ces figures.

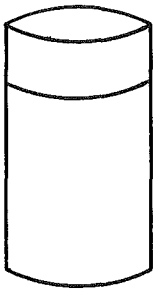
Début ou fin du programme

Lecture ou écriture
d'une carte mécanographique

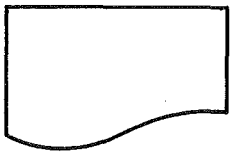
Lecture de ruban perforé



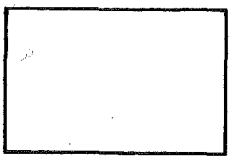
Lecture ou écriture sur une bande magnétique



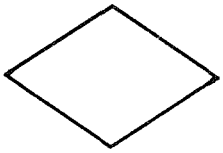
Lecture ou écriture sur des disques magnétiques



Sortie des résultats sur imprimante



Opération interne (Addition, soustraction ... transmission)



Condition

Dans un ordinogramme, les différentes figures sont reliées entre elles par un trait. Lorsque la lecture se fait dans le sens naturel, soit de la gauche vers la droite ou du haut vers le bas, aucune flèche ne figure sur les traits. Par contre, le sens du trait sera indiqué par une flèche si l'ordinogramme doit être lu de la droite vers la gauche ou du bas vers le haut.

Nous ferons à présent quelques exercices afin d'examiner plus concrètement les possibilités qu'offre la technique des ordinogrammes.

Exercice 1

Ce premier exercice consiste à dessiner un ordinogramme destiné à lire chacune des cartes d'un fichier, à compter ces cartes et à en imprimer le contenu. Puis, en fin de travail à imprimer le nombre total de cartes lues. Le programme devra effectuer les opérations suivantes :

a) Lecture du fichier de cartes perforées; l'ordinateur lit une carte, exécute les traitements prévus et revient ensuite à l'opération de lecture. Il est nécessaire de *déclarer une zone de lecture (ZL) de 80 positions* dans laquelle le contenu d'une carte sera stocké. Il faut remarquer qu'il n'y a jamais qu'une seule carte dans l'unité centrale.

b) Impression des résultats; après chaque lecture, on trouve une opération d'écriture sur l'imprimante. Cette opération pourra se faire en utilisant la zone de lecture (ZL).

c) Totalisation du nombre de cartes lues; pour effectuer ce dénombrement, il faut déclarer *une zone-compteur (X)* qui au début du programme vaudra 0 (zéro) et à laquelle on ajoutera 1 après chaque lecture.

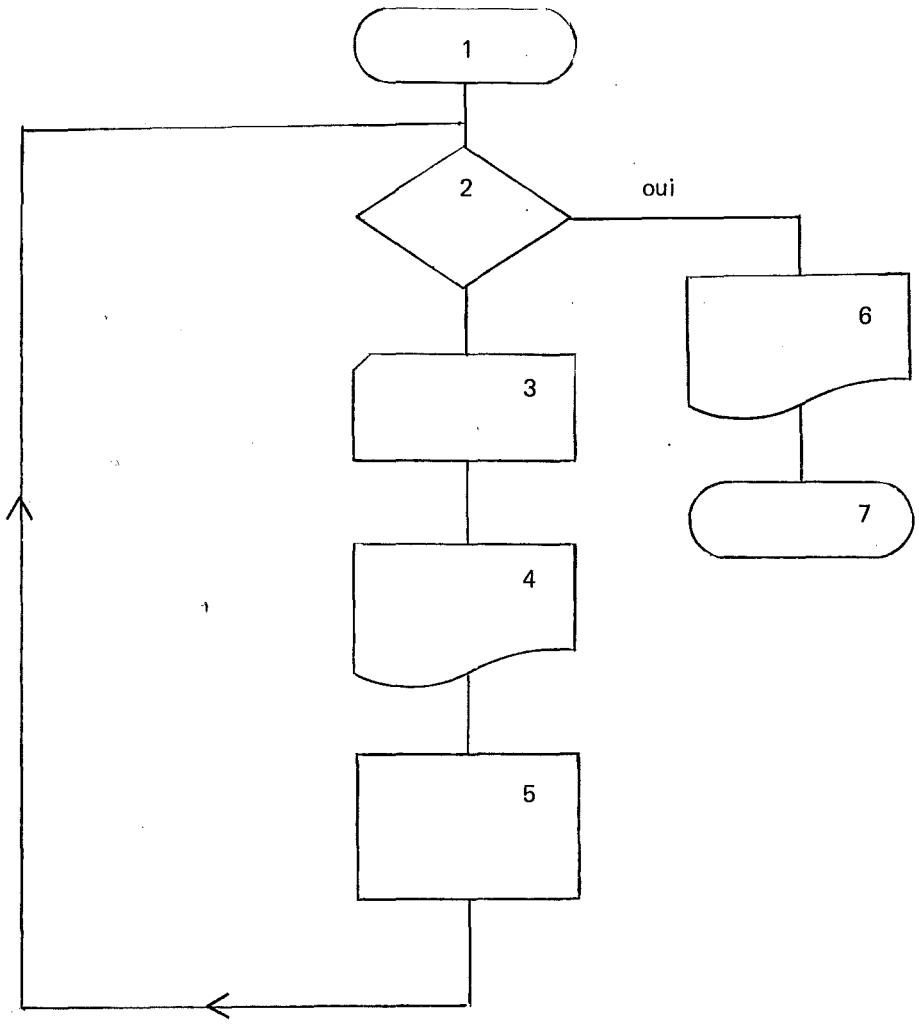
Ces trois opérations seront répétitives, c'est-à-dire, que l'ordinateur les exécutera pour chacune des cartes du fichier. Dès lors, il faut prévoir après l'ordre C.

d) Un ordre de saut inconditionnel vers le début du programme et cela jusqu'au traitement de la dernière carte.

e) Après avoir lu, imprimé et comptabilisé la dernière carte, l'ordinateur passe aux instructions finales du programme où il effectue, avant *l'ordre d'arrêt*, un *ordre d'impression du contenu de la zone-compteur (X).*

Il y aura donc dans le programme *deux déclarations de zone*, ZL, la zone de lecture-écriture et X, le compteur où les données seront dénombrées. Ces deux déclarations n'apparaissent pas dans l'ordinogramme.

L'ordinogramme



Commentaires

1) Début du programme.

2) Cet ordre est une condition; on pose la question suivante : est-ce que la dernière carte du fichier a été lue ? Si c'est le cas, on passe à l'opération 6, autrement on exécute l'ordre 3. Ce test de fin de fichier se retrouve dans tous les programmes qui font appel à un fichier d'entrée. Généralement cette condition se place en début d'ordinogramme.

3) Lecture d'une carte dans la zone ZL. Cet ordre fait passer dans l'unité de traitement les informations contenues dans une carte. Cette instruction détruit le contenu antérieur de la zone ZL en y plaçant le contenu de la carte lue. Je rappelle, il convient d'y insister, qu'il n'y a jamais dans l'unité centrale que le contenu d'une seule carte.

4) Ecriture sur l'imprimante du contenu de ZL.

5) Totalisation dans le compteur X de la carte traitée. Chaque fois que l'ordinateur exécute cet ordre, il ajoute 1 au compteur X. Après avoir effectué l'ordre 5, on remonte à l'instruction 2 pour reprendre le processus complet de traitement d'une carte, c'est l'aspect répétitif d'un programme.

6) Cet ordre sera effectué après détection de la fin du fichier, c'est-à-dire, après le traitement de la dernière carte. A ce stade, l'ordinateur imprime le contenu de X qui équivaut au nombre de cartes traitées.

7) Instruction de fin du programme : arrêt.

Trois instructions sur sept ne sont pas répétitives, ce sont les ordres 1, 6 et 7. Le premier et le dernier sont respectivement le début et la fin du programme, il est normal qu'ils ne soient exécutés qu'une seule fois. En ce qui concerne l'ordre 6, on comprend facilement qu'il se place après le test de fin de fichier et qu'il ne soit pas répétitif puisqu'il consiste à imprimer le nombre total de données traitées : cela ne peut se faire qu'après la lecture intégrale du fichier, donc à la fin des opérations.

Exercice 2

Dans les pages qui suivent, je voudrais montrer qu'il n'existe pas de solution standard pour traiter un problème déterminé. Le choix de la méthode est souvent une question de bons sens, d'équilibre entre les moyens à mettre en oeuvre. Ceux-ci sont déterminés par la complexité du problème, la durée du traitement, la longueur du fichier et le nombre de variables dans le fichier.

Imaginons le problème suivant pour lequel je propose quatre solutions : on doit traiter un fichier de plusieurs milliers de données enregistrées sur cartes mécanographiques. Dans la colonne 1 des cartes se trouve un code symbolique représenté par des chiffres qui vont de 1 à 6. Il faut compter le nombre de fois que chaque code apparaît dans le fichier.

Solution 1

Cette solution consiste à lire les données dans une zone de lecture ZL et à effectuer six interrogations successives pour voir si la zone de lecture contient 1, 2, 3, 4, 5 ou 6.

Puisque le but du programme est de dénombrer les apparitions de chacun des six codes, on doit déclarer au préalable un compteur pour chaque code, soit 6 compteurs que nous appellerons XA, XB, XC, XD, XE et XF. XA sera destiné à totaliser les occurrences du code 1, XB celles du code 2, XC celles du code 3, etc. Voici les principales étapes du programme.

a) Test de fin de fichier.

b) Lecture d'une carte. Je rappelle que l'on a déclaré une *zone de lecture (ZL)*.

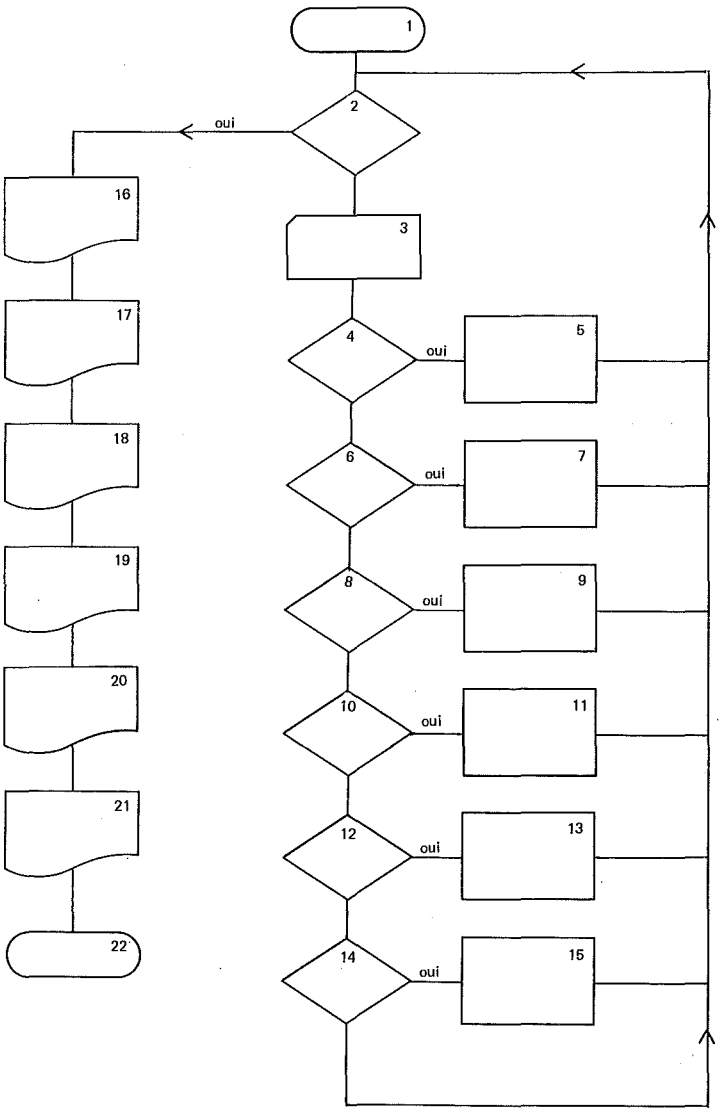
c) Première interrogation sur le contenu de la colonne 1 de ZL : le code lu dans ZL est-il 1 ? Si la réponse à cette question est oui, l'ordinateur ajoute la valeur 1 au compteur XA.

Le programme contiendra en tout 6 interrogations semblables à celle-ci et six opérations de comptage. Chaque interrogation et chaque comptage concerne une des modalités de la zone ZL.

d) La fin du programme consiste à imprimer les valeurs contenues dans chacun des six compteurs, c'est-à-dire le nombre d'apparitions de chaque code.

Pour cette première solution, sept déclarations sont nécessaires : une zone de lecture ZL et six compteurs destinés à dénombrer chacun une des modalités de la variable.

L'ordinogramme



Commentaires

1) Début du programme.

2) Test de fin de fichier. Si la dernière carte a été traitée, l'ordinateur exécute l'opération 16; autrement, il passe à l'instruction 3.

3) Lecture d'une carte dans la zone ZL.

4) Interrogation sur le contenu de la colonne 1 de la zone ZL. Si cette colonne contient le code 1, l'ordinateur exécute l'opération 5; dans le cas contraire, il va à l'opération 6.

5) Ajouter 1 dans le compteur XA. Cette addition permet de compter les apparitions du code 1 dans le fichier. Après l'opération 5, l'ordinateur retourne à 2.

6, 8 10, 12 et 14) Ces ordres sont des interrogations sur le contenu de ZL : ils ont pour but de vérifier si ZL contient respectivement 2, 3, 4, 5 ou 6. Lorsque l'ordinateur détecte un de ces codes, il passe à l'instruction de comptage qui correspond à l'interrogation pour laquelle il a obtenu une réponse positive.

7, 9, 11, 13 et 15) Augmenter de la valeur 1 un des compteurs XB, XC, XD, XE ou XF selon que l'on repère à une des opérations 6, 8, 10, 12 ou 14, le code 2, 3, 4, 5 ou 6.

14) A la suite de cette instruction, l'ordinateur retourne à l'opération 2 dans le cas où le code n'est pas 6. Cela se produit lorsqu'on a dans ZL un code inférieur à 1 ou supérieur à 6, c'est-à-dire un code qui n'est pas prévu.

16) Après la détection de la fin du fichier à la figure 2, l'ordinateur va à l'instruction 16 qui consiste à imprimer le contenu du compteur XA, c'est-à-dire le nombre de cartes qui portent le chiffre 1.

17, 18, 19, 20 et 21) Ces instructions sont destinées à imprimer le contenu des compteurs XB, XC, XD, XE et XF.

22) Après l'impression en 21 du nombre de cartes codées 6, l'ordre d'arrêt est rencontré.

Cet ordinogramme appelle deux remarques.

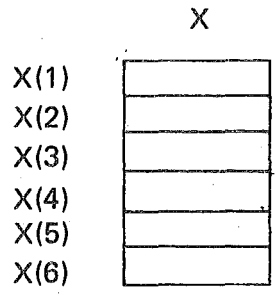
a) La répétition d'instructions identiques est évidente (4, 6, 8, 10, 12 et 14, d'une part, 5, 7, 9, 11, 13 et 15, d'autre part, et enfin, 16, 17, 18, 19, 20 et 21 sont des ordres semblables). Cette répétition résulte d'une logique simple et purement linéaire.

b) Le programme que l'on pourrait écrire à partir de cet ordinogramme serait correct et probablement très performant en raison même de la simplicité de la logique. Cependant, on fera sans doute à cet ordinogramme et au programme le reproche d'être trop longs et de répéter les mêmes opérations. Dès lors, on recherchera des solutions plus brèves.

Solution 2

Cette solution consiste essentiellement à utiliser une *table*, c'est-à-dire une zone qui est composée de zones plus petites que l'on appelle *les éléments de la table.* Ces éléments qui sont de même type et généralement de même longueur peuvent être alphabétiques ou numériques. Un élément de table présente deux informations :

1) *Un numéro qui indique sa position dans la table.* Ainsi, dans une table X de 6 éléments, on distingue X(1), X(2), X(3), X(4), X(5) et X(6). Concrètement, on pourrait représenter la table X sous cette forme :



L'ensemble de la figure représente la table X. Chaque ligne est un élément de cette table. Le chiffre qui est placé entre parenthèses à droite du nom de la table et qui indique le numéro d'ordre de l'élément dans la table s'appelle un *indice*. Les chiffres qui sont ici des *indices constants* peuvent être remplacés par un *indice variable*, c'est-à-dire par le nom d'un compteur - que nous appellerons J - auquel on attribue au départ du programme telle ou telle valeur. Cette valeur initiale peut être modifiée selon les nécessités du traitement au cours du déroulement du programme.

Supposons que l'indice-compteur qui permet de désigner un des éléments de la table X s'appelle J. Si on donne à J la valeur 1, l'expression X(J) désignera l'élément 1 de X. Par ailleurs, si à un autre moment du programme, on attribue à J la valeur 4, par exemple, alors l'expression X(J) désignera le quatrième élément de la table X.

2) La seconde information que présente un élément de table est une donnée, c'est-à-dire son propre contenu. Comme je l'ai dit précédemment, cette information peut être numérique ou alphabétique. Dans le cas de données numériques, la table sera en réalité composée d'éléments-compteurs. Ainsi, la table X se présentera comme suit :

La table X a été déclarée comme une zone numérique de 6 éléments. Dès lors, chaque élément de X peut servir de compteur. L'exemple ci-contre montre qu'après un travail déterminé, l'élément 1 de X vaut 21, l'élément 2 vaut 0, l'élément 3 vaut 5, etc.

X(1)	21
X(2)	0
X(3)	5
X(4)	17
X(5)	9
X(6)	10

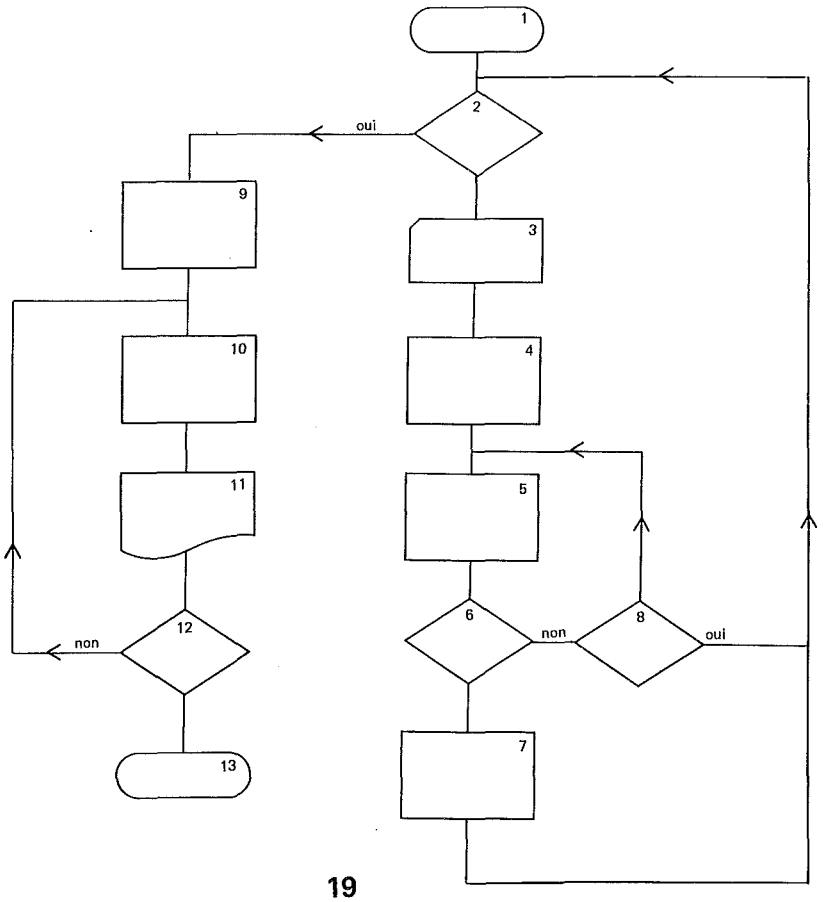
Dans le cas qui nous intéresse du dénombrement d'une variable à six modalités, chaque élément d'une table numérique à six parties servira à dénombrer les apparitions d'une des modalités de la variable : l'élément X(1) sera utilisé pour compter le nombre de cartes qui portent le chiffre 1 en colonne 1, l'élément X(2) comptera les cartes qui sont codées 2. L'indice J aura une double fonction. Non seulement il servira de compteur et indiquera le numéro de l'élément de la table X dans lequel on veut effectuer une opération, mais, en outre, il servira d'élément de comparaison. Il sera comparé au contenu de la zone de lecture. Lorsque les deux zones, l'indice J et la zone de lecture, seront identiques, le programme ajoutera la valeur 1 au nombre figurant dans l'élément de la table dont J fournit le numéro. L'exemple suivant éclairera le processus : supposons que la zone de lecture contienne un code 3. L'indice-compteur J que l'on fera varier de 0 à 6 par additions successives de 1, atteindra, à un moment donné, la valeur 3. Il y aura alors égalité entre le contenu de la zone de lecture et la valeur de l'indice J. Cette égalité conduira l'ordinateur à ajouter 1 au nombre contenu dans l'élément J de X, c'est-à-dire dans l'élément 3 (*X(3)*).

Les étapes principales de la deuxième solution seront :
1) Test de fin de fichier.
2) Lecture d'une carte. Cette opération nécessite la déclaration d'une zone de lecture (ZL).

3) Initialisation d'un indice à zéro (J). Cet indice variera de 1 à 6 jusqu'à ce que son contenu soit identique à la donnée lue dans ZL.
4) Lorsqu'il y aura identité entre ZL et J, l'ordinateur effectuera une addition de 1 dans l'élément J de la table X que l'on aura déclarée préalablement.
5) Impression des résultats.

Les déclarations seront donc la zone de lecture ZL, la table X composée de six éléments-compteurs et l'indice J.

L'ordinogramme



Commentaires

1) Début du programme.

2) Test de fin de fichier. Après le traitement de la dernière carte, l'ordinateur saute à l'instruction 9.

3) Lecture d'une carte dans la zone ZL.

4) Initialisation à 0 de l'indice J qui sera comparé au contenu de ZL et qui servira à désigner l'élément de X auquel on ajoute 1 à un moment donné.

5) Addition de 1 à J. Cette opération a pour but de faire varier J. Lorsque l'on vient de la figure 4, l'opération 5 revient à faire J = 0 + 1.

6) Cette comparaison a pour but de vérifier s'il y a identité entre le contenu de la colonne 1 de ZL et la valeur de l'indice. Si les deux zones sont égales, on va à l'opération 7.

7) Addition de 1 dans l'élément J de X. X(J) = X(J) + 1. Cet ordre qui s'effectue lorsqu'il y a identité entre la valeur de l'indice et le contenu de ZL, sert à compter le nombre de fois que l'on rencontre chacun des codes dans le fichier. Après avoir exécuté cette opération, l'ordinateur saute à l'instruction 2.

8) Cet ordre conditionnel vérifie si l'indice vaut 6. Si c'est le cas, l'ordinateur passe à l'instruction 2, autrement l'instruction 5 est répétée. Voici le rôle de l'opération 8. La colonne 1 de ZL contient une variable à 6 modalités; dès lors J doit varier de 1 à 6 pour prendre une valeur égale à chacun des codes possibles. Lorsque J atteint la valeur 6, et que l'on se trouve à l'instruction 8, on a déjà comparé la donnée contenue dans ZL aux différentes valeurs qu'elle peut avoir (la comparaison se fait à l'instruction 6). Par conséquent, le traitement pour la carte qui se trouve dans ZL est terminé puisque l'on n'a pas dans ZL un des codes de 1 à 6. En effet, l'identité entre ZL et J aurait provoqué le saut à l'instruction 7 et le retour à 2.

9) Après détection de la fin du fichier, le contrôle est donné à l'ordre 9 où débute la partie du programme consacrée à l'impression des résultats. Cette opération initialise l'indice J à 0.

10) Addition de 1 à J. J = J + 1.

11) Impression de l'élément J de X. La valeur contenue dans un élément de X donne le nombre d'apparitions de la modalité J de la variable.

12) Si J est inférieur à 6, on revient à l'ordre 10 ajouter 1 dans J. Si J vaut 6 c'est-à-dire si l'on a imprimé tous les éléments de la table X, le contrôle est donné à l'opération 13.

13) Arrêt.

Il n'est peut-être pas inutile de s'attarder encore un peu à cette solution en montrant comment, pour une donnée précise, l'ordinateur effectue le travail. Prenons, par exemple, une carte qui contient en colonne 1 le code 3 et voyons ce que fait l'ordinateur.

3) Lecture d'une donnée dans ZL.
4) Mise à 0 de J — J = 0
5) J = J + 1 — J = 1
6) Est-ce que J (égal à 1) est égal à la position 1 de ZL (=3) ?
La réponse est non donc on passe à 8.
8) J vaut-il 6 ? La réponse est non, on va à 5.
5) J = J + 1 — J = 2
6) J (=2) est-il égal à ZL (=3) ? Non, l'ordinateur répète l'instruction 8.
8) Vaut-il 6 ? Non, saut à 5.
5) J = J + 1 — J = 3
6) J est-il égal à ZL ? Cette fois la réponse est positive; dès lors, on exécute l'ordre 7.
7) X(J) = X(J) + 1. L'élément de X qui correspond à la modalité de la variable que l'on traite s'accroît de 1. Dans ce cas c'est le troisième élément de X qui est augmenté.

Cette deuxième solution, plus complexe certes que la première, présente l'avantage

d'être beaucoup plus courte puisqu'elle ne comporte que 13 instructions contre 22 pour la précédente.

Solution 3

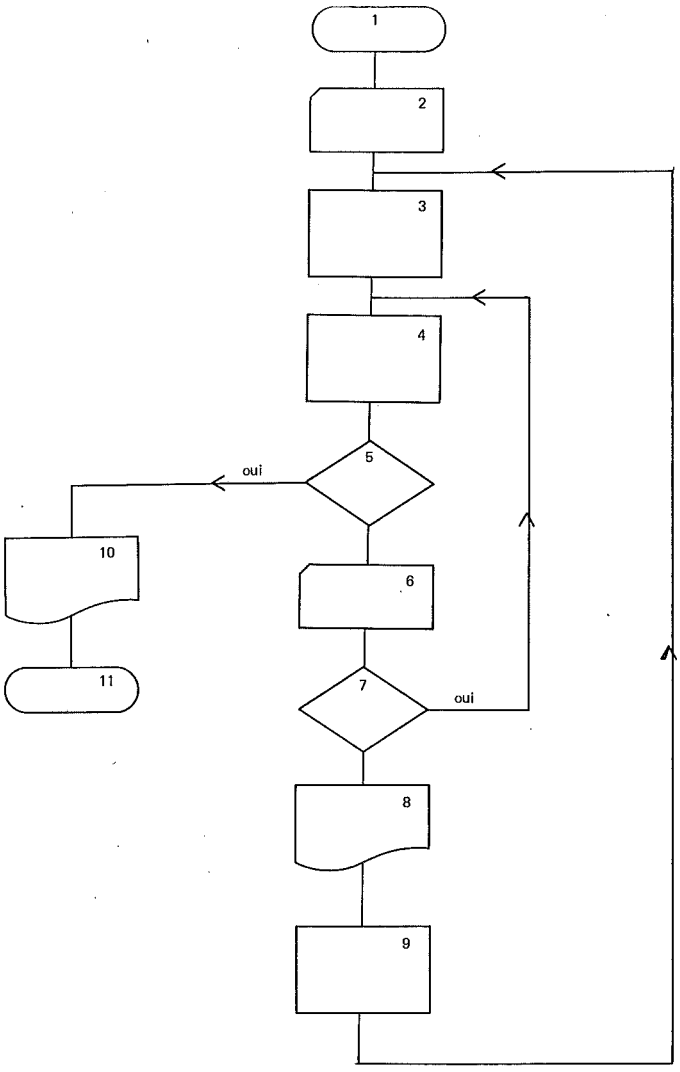
Cette troisième manière d'effectuer des dénombrements diffère des précédentes pour la raison suivante : au lieu de lire une carte et de s'interroger sur son contenu, on compare simplement les données deux à deux du début à la fin du fichier. Pour cela, les données doivent être rangées en ordre croissant ou décroissant avant le traitement proprement dit de sorte que les informations identiques soient physiquement regroupées dans le fichier. Si, on prend, par exemple, un fichier de 12 cartes, les données doivent se présenter comme suit :

Carte n°	*Contenu*	*Carte n°*	*Contenu*
1	1	7	4
2	1	8	4
3	1	9	4
4	2	10	5
5	2	11	6
6	3	12	6

Le procédé de comptage est simple. Il consiste à utiliser une zone de lecture ZB et une zone de stockage (ZA) de la donnée antérieure à celle que l'on a lue dans ZB. Cette zone de stockage ZA servira en outre à lire la première donnée physique du fichier. L'instruction principale du programme sera une comparaison entre le contenu de ZA et celui de ZB. Or, puisque les données identiques ont été regroupées, il n'y aura de différence entre ZA et ZB que lorsque toutes les cartes portant une même variable auront été traitées. Comme on le voit cette méthode se fonde sur le manque d'identité entre deux zones. Les étapes principales du programme seront les suivantes :

1) Lecture de la première donnée du fichier dans la zone ZA.
2) Initialisation d'un compteur X à 0. Ce compteur X est destiné à dénombrer les apparitions de chacune des modalités de la variable.
3) Lecture d'une carte dans la zone ZB. Cette opération de lecture sera utilisée pour toutes les données sauf pour la première.
4) S'il y a identité entre ZA et ZB, on ajoute 1 à la valeur de X.

L'ordinogramme



Commentaires

1) Début du programme.
2) Lecture de la première donnée du fichier dans la zone ZA. Cet ordre est exécuté une seule fois, l'ordinateur n'y revient pas. Il a pour but de garnir une première fois la zone ZA qui doit toujours contenir la donnée physiquement antérieure à celle que l'on lit dans ZB.
3) Initialisation à 0 du compteur X.
4) Addition de 1 dans le compteur X. Cette instruction permet de dénombrer les occurrences d'une variable.
5) Test de fin de fichier. Après le traitement de la dernière carte, l'ordinateur passe à l'ordre 10, autrement il exécute l'opération 6.
6) Lecture d'une carte dans la zone ZB. Lorsque l'ordinateur rencontre cette instruction pour la première fois, il lit la carte 2 puisque la carte 1 a déjà été lue dans la zone ZA à la figure 2.
7) L'ordre 7 vérifie s'il y a identité entre le contenu de ZA et la donnée lue dans ZB. Si c'est le cas l'ordinateur saute à l'ordre 4, sinon il passe à l'opération 8.
8) Impression de la zone ZA et de la valeur de X. On imprime ici la modalité de la variable et sa fréquence d'apparition.
9) Le contenu de la zone ZB est transporté dans la zone ZA. Après cette opération, l'ordinateur remonte à la figure 3.
10) Impression du contenu de ZA et de la valeur de X. La dernière modalité de la variable et sa fréquence sont imprimées après la détection de la fin du fichier.
11) Arrêt.

Pour terminer l'examen de cette troisième solution, nous allons faire dénombrer le fichier de 12 cartes que nous avons vu précédemment.

1) Début du programme.	
2) Lecture dans ZA de la carte 1.	ZA contient 1
3) Initialisation de X à 0.	X = 0
4) Addition de 1 à X.	X = 1

5) Test de fin de fichier.	
6) Lecture de la deuxième carte dans ZB	ZB contient 1
7) Il y a identité entre ZA et ZB	retour à 4
4) Addition de 1 à X.	X = 2
5) Test de fin de fichier.	
6) Lecture de la troisième carte dans ZB	ZB contient 1
7) Il y a identité entre ZA et ZB	retour à 4
4) Addition de 1 à X.	X = 3
5) Test de fin de fichier.	
6) Lecture de la quatrième carte dans ZB.	ZB contient 2
7) ZA et ZB sont différents, l'ordinateur va à l'ordre 8.	
8) Impression de ZA et de X; on imprime .....1.....3 ce qui signifie que le code 1 apparaît trois fois dans le fichier.	
9) On transporte dans ZA le contenu de ZB. On retourne ensuite à l'ordre 3 pour reprendre l'ensemble du processus.	ZA contient 2

Cette troisième solution est à bien des égards la meilleure : elle est la plus brève, donc elle occupe peu de place dans l'unité centrale de traitement. La logique est simple, dès lors elle permet un travail rapide. Enfin, ce procédé permet de dénombrer les modalités d'une variable quel que soit leur nombre. Le seul inconvénient de cette solution est qu'elle impose un tri préalable du fichier.

Solution 4

Cette quatrième méthode peut être considérée comme une généralisation du procédé de comptage utilisé pour la solution 2. Elle consiste à déclarer *deux tables* dans l'unité centrale. *La première sert à dénombrer les modalités* d'une variable; il s'agit donc, comme dans la solution 2, d'une table numérique composée d'éléments-compteurs. *La deuxième table est utilisée pour stocker les codes* représentant les diverses modalités de la variable étudiée; ces codes seront donnés à l'ordinateur au départ du programme afin d'être enregistrés dans la table à raison d'un code par élément. Après avoir reçu les codes, l'ordinateur pourra traiter les données en comparant chaque élément de la table

des codes à la donnée lue; il s'agit dans ce cas d'une table d'éléments comparateurs. Lorsqu'il y a identité entre une donnée et un des éléments de la table des codes, l'ordinateur ajoute la valeur 1 à l'élément de la table numérique qui porte le même numéro que l'élément de la table des codes identique à la donnée lue. Ce procédé présente deux avantages.

1) Le même programme peut être employé pour traiter des variables codées de manière différente.

2) Le nombre de codes peut différer d'une variable à l'autre. C'est au moment où l'on donne la liste des codes à l'ordinateur que celui-ci calcule le nombre de modalités.

Prenons deux exemples de comptages à faire.

Dans le premier cas, les codes sont représentés par les chiffres de 1 à 9. Ces chiffres, perforés sur cartes, sont lus par l'ordinateur qui les range dans la table à raison d'un code par élément. Après cette opération de stockage, l'ordinateur va exploiter le fichier : il compare chaque donnée lue à chacun des éléments de la table. Supposons que l'ordinateur ait lu une carte qui porte le chiffre 4; puisque le code 4 a été enregistré dans le quatrième élément de la table des codes, il y aura identité entre cet élément de table et la donnée lue, c'est pourquoi l'ordinateur ajoutera 1 à l'élément-compteur 4 de la table numérique.

Dans le second exemple, les codes sont représentés par les dix premières lettres de l'alphabet de A à J. Ces lettres sont stockées dans la table en respectant l'ordre alphabétique; ainsi, la lettre E, cinquième lettre de l'alphabet, occupera l'élément 5 de la table des codes. Supposons maintenant que l'ordinateur lise une carte dans laquelle on a perforé la lettre E. Il y aura identité entre le contenu de la carte et l'élément 5 de la table des codes et, par conséquent, le programme ajoutera la valeur 1 à l'élément-compteur 5 de la table numérique.

La déclaration de la table des codes doit viser à améliorer la souplesse du programme. D'une part, il faut que le nombre d'éléments de la table soit suffisant pour contenir des codes relativement nombreux. Ainsi, si une variable a cinquante modalités, la table des codes devra se composer de 50 éléments. D'autre part, il est préférable de déclarer une table alphanumérique. En effet, une table numérique ne peut contenir que les dix chiffres tandis qu'une table alphanumérique pourra recevoir toutes les lettres de l'alphabet, tous les chiffres et tous les caractères spéciaux.

Le programme se composera de trois groupes d'instructions :

a) Enregistrement des codes à partir d'un petit fichier de cartes perforées. Les codes seront stockés dans une table alphanumérique. On doit déclarer à ce niveau *une zone de lecture (ZL)* et *une table de stockage des codes (TA)* dont le nombre d'éléments sera 100. En outre, on utilisera *un indice L* qui désignera l'élément de TA dans lequel on doit ranger les différents codes. Cet indice sera initialisé à 0 au départ du programme et sera augmenté de 1 après chaque opération de lecture. Lorsque tout le fichier des codes aura été lu, l'indice L aura une valeur égale au nombre de codes enregistrés dans la table TA.

b) Traitement des données : cette partie du programme effectue le comptage des modalités de la variable. On trouvera ici une structure de programme identique à la solution 2. Les données seront lues dans une zone qui pourra être la même que celle qui a été déclarée dans la première partie du programme (ZL). On déclarera *une table numérique de 100 éléments-compteurs (X)* et on utilisera *un indice variable (J)* qui aura le même rôle que l'indice J employé dans la solution 2.

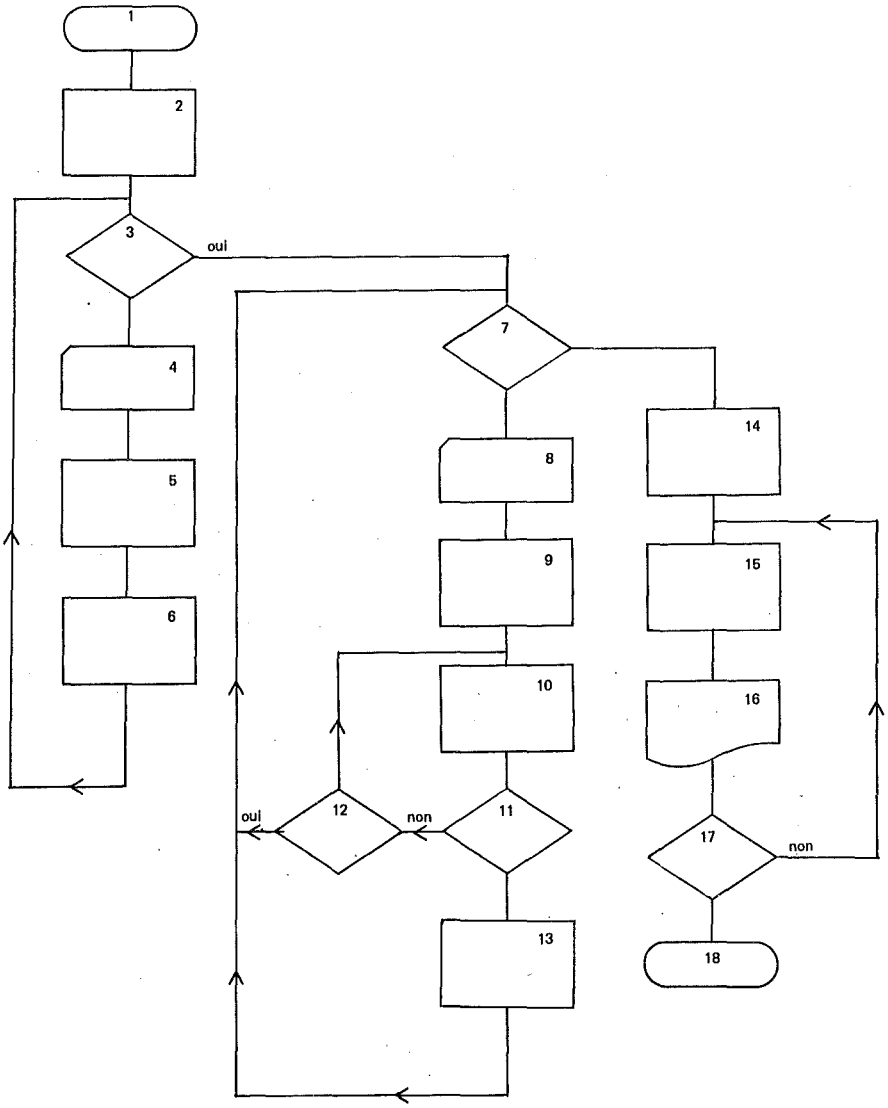
c) Impression des résultats : pour cette étape également, le traitement sera semblable à celui que j'ai exposé dans la deuxième solution.

Trois déclarations sont nécessaires dans cette solution :

1) Une zone de lecture (ZL) pour le fichier des codes et pour le fichier des données;
2) Une table alphanumérique (TA) pour le stockage des codes;
3) Une table numérique (X) pour le comptage de la fréquence de chacun des codes.

En outre, on rencontrera dans le programme deux indices, L et J, qui serviront à accéder aux éléments des tables TA et X.

L'ordinogramme



Commentaires

1) Début de programme.

2) Initialisation de l'indice L à 0. Cet indice contient le numéro de l'élément de la table TA où il faut ranger un code. Ainsi, lorsque L vaut 2, l'ordinateur rangera le code lu dans ZL, dans l'élément 2 de TA. L aura un autre rôle : après la lecture du fichier des codes, la valeur de L sera égale au nombre de cartes lues, c'est-à-dire au nombre de codes. Supposons que l'ordinateur ait lu dix cartes et enregistré les codes dans les éléments 1 à 10 de TA. A ce moment le fichier des codes est entièrement lu et L a une valeur qui correspond au numéro du dernier élément de TA où l'on a stocké une information; L vaut donc 10.

3) Test de fin du fichier des cartes-codes. Après avoir traité la dernière carte, l'ordinateur va à l'ordre 7, autrement il exécute l'opération 4.

4) Lecture d'une carte dans la zone ZL. Chaque carte contient un code.

5) Addition de 1 à l'indice L. Cette opération a pour but de donner à L une valeur égale au numéro de l'élément de TA où l'on doit ranger le code lu dans ZL. Au moment où l'ordinateur lit le premier code du fichier, L vaut 1 et le code est rangé dans l'élément 1 de TA.

6) Enregistrement dans l'élément L de TA de la donnée lue en 4. Cette opération consiste à transférer dans un élément de TA le code contenu dans ZL.

Ces six opérations constituent la première partie du programme durant laquelle les codes perforés sur cartes sont enregistrés dans la table TA. Imaginons que l'on ait à traiter une donnée qui peut être représentée par cinq codes, les lettres de A à E, au terme de cette première partie la table TA se présentera comme suit :

TA(1)	A
TA(2)	B
TA(3)	C
TA(4)	D
TA(5)	E

A ce moment du programme, l'indice L vaut 5 ce qui signifie que l'ordinateur a enregistré 5 codes.

Les instructions 7 à 13 constituent la deuxième partie du programme. Durant cette partie, l'ordinateur doit lire les données et compter la fréquence d'apparition des différents codes.

7) Test de fin de fichier. Lorsque la fin du fichier des données est détectée, l'ordinateur saute à l'ordre 14, autrement il exécute l'opération 8.

8) Lecture d'une donnée dans la zone ZL.

9) Initialisation à 0 de l'indice J. Cet indice variera de 0 à L pour permettre de comparer chaque élément de la table-code TA au contenu de ZL. Lorsque J aura une valeur égale à celle de L cela signifiera que l'ordinateur a effectué autant de comparaisons qu'il y a de codes dans la table TA puisque L a notamment servi dans la première partie du programme à compter le nombre de codes stockés dans TA.

10) Addition de 1 à l'indice J.

11) Cette opération vérifie si le contenu de la zone ZL est égal au contenu de l'élément J de TA. Si c'est le cas, l'ordinateur exécute l'ordre 13, autrement il saute à l'opération 12.

12) Comparaison entre les valeurs de J et de L. Si J a atteint la valeur de L, l'ordinateur remonte à l'ordre 7, s'il n'y a pas égalité entre les deux indices, il va à l'opération 10. Les indices L et J seront égaux à l'ordre 12 lorsque ZL contient un code autre que ceux qui sont dans TA.

13) Addition de 1 à la valeur de l'élément J de X. Cette addition sert à compter une donnée identique au code contenu dans l'élément J de TA. Après avoir effectué l'ordre 13, l'ordinateur saute à l'instruction 7.

La troisième partie du programme est destinée à l'impression des résultats.

14) Initialisation à 0 de l'indice J. J va à nouveau varier de 0 à L.

15) Addition de 1 à la valeur de J.

16) Impression de l'élément J de X.

17) Si J et L sont égaux l'ordinateur va à l'ordre 18, autrement il remonte à 15.

18) Fin du programme.

Cette solution présente à peu près les mêmes avantages que la précédente puisque les modalités de la variable peuvent être en nombre indéterminé. Au surplus, elle permet de laisser le fichier dans son état original, ce qui dans certaines circonstances est indispensable.

En revanche, cette quatrième solution produira un programme moins performant que la troisième méthode surtout si le nombre de codes est très élevé.

*

* *

La deuxième partie de ce chapitre consacré à l'ordinogramme envisage le traitement des données linguistiques. La principale difficulté que l'on rencontre dans ce cas résulte de la longueur variable des informations. En effet, les mots, les phrases et les textes sont de longueurs inégales et lorsque l'on veut, au cours d'un traitement, retrouver un mot déterminé, il faut commencer par repérer ce mot en l'isolant du reste du discours.

Exercice 3

Le premier exemple de traitement d'informations linguistiques consistera à rechercher dans un texte les apparitions d'un mot déterminé.

Le texte a été perforé dans les colonnes 1 à 75 des cartes mécanographiques. Entre chaque mot, il y a, soit un blanc typographique, soit un signe de ponctuation. En outre, il a été convenu qu'un mot se termine toujours sur la carte où il commence et que, par conséquent, il ne peut y avoir de trait d'union pour marquer la continuation d'un mot sur la carte suivante.

Les étapes du programme seront les suivantes :
a) Après la lecture d'une carte dans une zone ZL, l'ordinateur va examiner cette zone position par position depuis le premier octet jusqu'au septante-cinquième pour rechercher la fin des mots. Chaque fois qu'une fin de mot sera détectée, l'ordinateur passera

à la suite du traitement. La dernière lettre d'un mot est toujours suivie, soit par un blanc, soit par une ponctuation; dès lors, lorsque l'ordinateur aura repéré, dans une position de ZL, un blanc ou une ponctuation, il saura qu'un mot se termine à la position précédente.

Pour effectuer la recherche d'un blanc ou d'une ponctuation, il est important de savoir que les signes de ponctuation, les caractères alphabétiques et les chiffres prennent dans l'unité centrale de traitement une *valeur binaire* qui permet de les *identifier et de les classer en ordre croissant*. C'est ainsi que le blanc et les signes de ponctuation sont plus petits que les lettres (la valeur binaire du blanc est moindre que la valeur binaire d'une lettre). Les lettres sont représentées par des suites binaires qui respectent l'ordre alphabétique, c'est ainsi que A est plus petit que B, lequel est plus petit que C, ... et que Y est plus petit que Z. Enfin les chiffres sont supérieurs aux lettres (0 est plus grand que Z) et leurs valeurs binaires respectent l'ordre numérique (0 est plus petit que 1, 1 est inférieur à 2, ..., 8 est plus petit que 9).

Dans la recherche de la fin d'un mot, la dernière lettre se repère lorsque l'on trouve dans une position une information dont la valeur binaire est *inférieure* à celle de la lettre A.

b) Après avoir détecté la fin d'un mot, l'ordinateur compare le mot isolé dans la zone ZL à celui que l'on recherche.

c) En cas d'égalité lors de l'opération b), l'ordinateur imprime le contenu de la zone ZL où se trouve une occurrence du mot recherché et son contexte immédiat.

Une déclaration sera nécessaire dans ce programme : une zone de lecture ZL. En outre, on utilisera deux indices pour parcourir la zone ZL de la position 1 à la position 75. L'indice L indiquera la position à laquelle un mot commence et l'indice J donnera la dernière position de ce mot.

Pour faire comprendre la façon dont l'ordinateur travaille, il n'est peut-être pas inutile de montrer pour une courte phrase quelles seront les valeurs des deux indices à chacune des étapes du traitement. Prenons la phrase :

Qui aime bien, châtie bien.

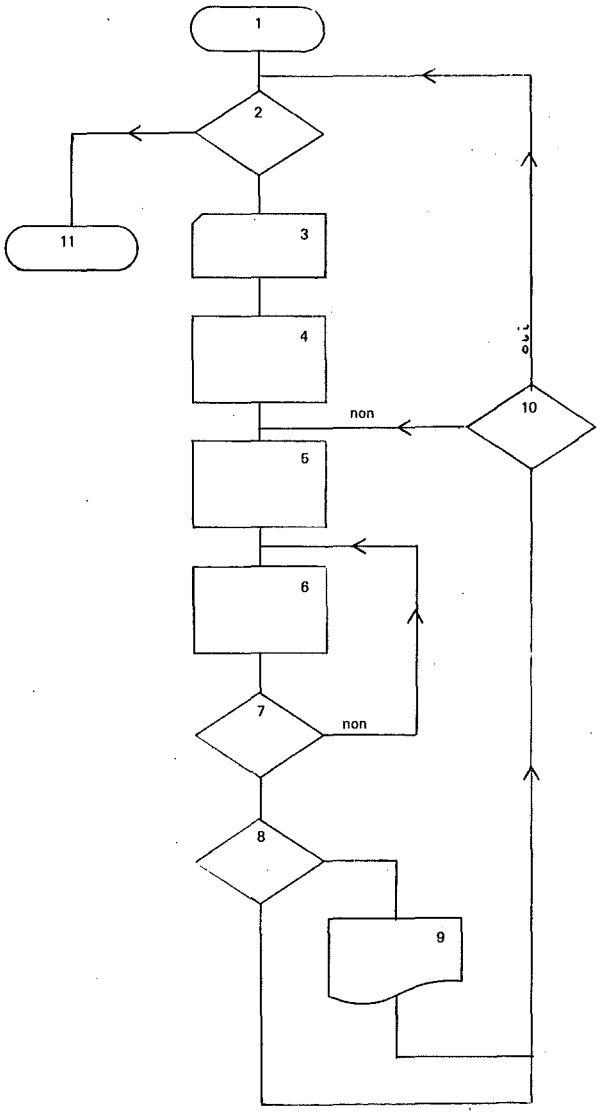
Cette phrase sera perforée sur une carte et lorsque l'ordinateur lira la carte, il placera son contenu dans la zone de lecture ZL. ZL se présentera comme le montre le schéma ci-dessous :

Si au départ du programme on attribue à l'indice J la valeur 0 et à l'indice L la valeur J + 1 soit 1, J ne désignera aucune position de ZL (la position 0 d'une zone n'existe pas) mais il sera préparé pour parcourir, position par position, la zone ZL. L, quant à lui, sera utilisé pour désigner la position où se trouve la première lettre du premier mot enregistré dans ZL.

L'initialisation des indices étant faite, on augmente l'indice J de la valeur 1 jusqu'à ce que la position J de ZL contienne une information inférieure à la lettre A. J vaudra 4 lorsque cette condition sera remplie; en effet, la position 4 de ZL contient un blanc. A ce moment l'ordinateur va contrôler si le mot qui commence à la position L de ZL, c'est-à-dire à la position 1, et qui se termine à la position J - 1, c'est-à-dire à la position trois, est le mot que l'on recherche. Après cela, le programme progresse dans l'examen de ZL en augmentant l'indice L de la valeur contenue dans J. L'addition L = J +1 donne comme résultat L = 4 +1 soit 5. L désigne à ce moment la position où se trouve la lettre initiale du mot à examiner.

Ce processus est répété jusqu'à la position 75 de la zone ZL. L prendra successivement les valeurs suivantes : 1, 5, 10, 15 et 22. J progressera de 0 à 75 et au moment où l'ordinateur vérifiera si le mot isolé dans ZL est le mot recherché l'indice J aura les différentes valeurs que voici : 4, 9, 14,21 et 26.

L'ordinogramme



Commentaires

1) Début du programme.

2) Test de fin de fichier. Après le traitement de la dernière carte, l'ordinateur va à l'instruction 11, autrement, il effectue l'opération 3.

3) Lecture d'une donnée dans ZL.

4) Initialisation à 0 de l'indice J. J sert à parcourir ZL position par position.

5) L'indice L est rendu égal à J et est simultanément augmenté de 1.

6) Addition de 1 à la valeur de J.

7) Cet ordre est une interrogation qui porte sur le contenu de la position J de la zone ZL. Si cette position contient une information inférieure à la lettre A (un blanc ou une ponctuation), cela signifie que l'ordinateur est arrivé à la fin d'un mot. Dans ce cas l'opération 8 est exécutée, autrement le programme saute à l'instruction 6.

8) La question que l'ordinateur rencontre ici est la suivante : est-ce-que le mot qui commence à la position L de ZL et qui se termine à la position J - 1 de cette zone est celui que l'on recherche ? Si c'est le cas, le programme va à l'opération 9, sinon il saute à l'ordre 10.

9) Impression du contenu de la zone ZL. Puisqu'on a repéré une occurrence du mot que l'on recherche, on imprime ce mot et son contexte. Après quoi l'ordinateur va à l'opération 10.

10) Cette opération vérifie si l'indice J vaut 76 auquel cas le traitement d'une carte est terminé puisque les mots sont perforés de la colonne 1 à la colonne 75. Si J est égal à 76, le programme rend le contrôle à l'ordre 2. Si, par contre, la valeur de l'indice J est inférieure à 76, on remonte à l'ordre 5 pour continuer le traitement de la donnée contenue dans ZL.

11) Fin du programme. L'ordinateur arrive à cette instruction après avoir traité tout le fichier.

Reprenons l'exemple de la phrase *Qui aime bien châtie bien*, dans laquelle on recherche l'apparition du mot *aime*, l'ordinateur va détecter une première fois un blanc au moment où la valeur de L est 1 et où J vaut 4. L'ordinateur passe alors à l'opération 8

et il vérifie si le contenu des positions 1 à 3 est le mot *aime*. Puisque ce n'est pas le cas, il retourne à l'opération 10 puis à l'instruction 5 pour accroître les valeurs de L et de J. Un deuxième blanc sera repéré à la position 9 de ZL alors que L vaut 5. A la question 8, l'ordinateur trouvera que le mot qui commence à la position 5 et qui se termine en 8 est le mot recherché.

Exercice 4

Ce deuxième problème concernant les informations linguistiques a pour but de préparer un fichier qui servira à constituer un *index inverse*. On doit traiter des cartes dans lesquelles les mots sont perforés dans les colonnes 1 à 20, à raison d'un mot par carte. Pour préparer un *index inverse*, il faut aligner les mots sur la gauche à partir de leur dernière lettre. On part donc d'une zone ZA dans laquelle les mots sont alignés sur la gauche et on transporte ces mots dans une zone ZB en les inversant. Ce passage d'une zone à l'autre se fait lettre par lettre en utilisant pour chacune des zones un indice, soit L pour la zone ZA et J pour la zone ZB.

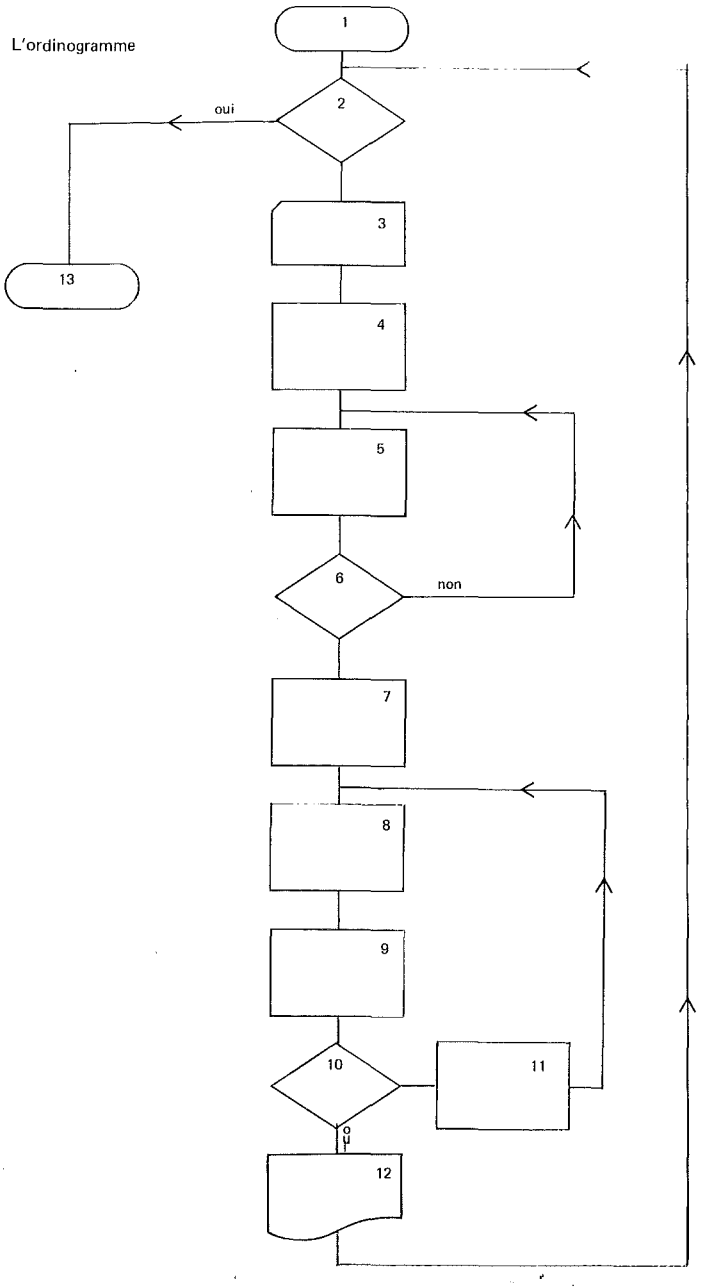
On trouvera dans les pages qui suivent deux solutions pour ce problème.

Solution 1

Le traitement se déroule en deux temps. Dans la première phase, la zone de lecture *ZA* est parcourue de gauche à droite jusqu'à ce que l'on rencontre la dernière lettre du mot identifiée par le fait qu'elle est suivie par une position blanche. La seconde phase consiste à transmettre dans la zone *ZB* le contenu de *ZA*. A ce stade, le contenu de *ZA* est transporté à partir de la dernière lettre du mot, lettre par lettre, dans *ZB*.

Deux déclarations sont nécessaires pour cette solution, la zone de lecture *ZA* et la zone dans laquelle on rangera le mot inversé que nous appellerons *ZB*.

L'ordinogramme



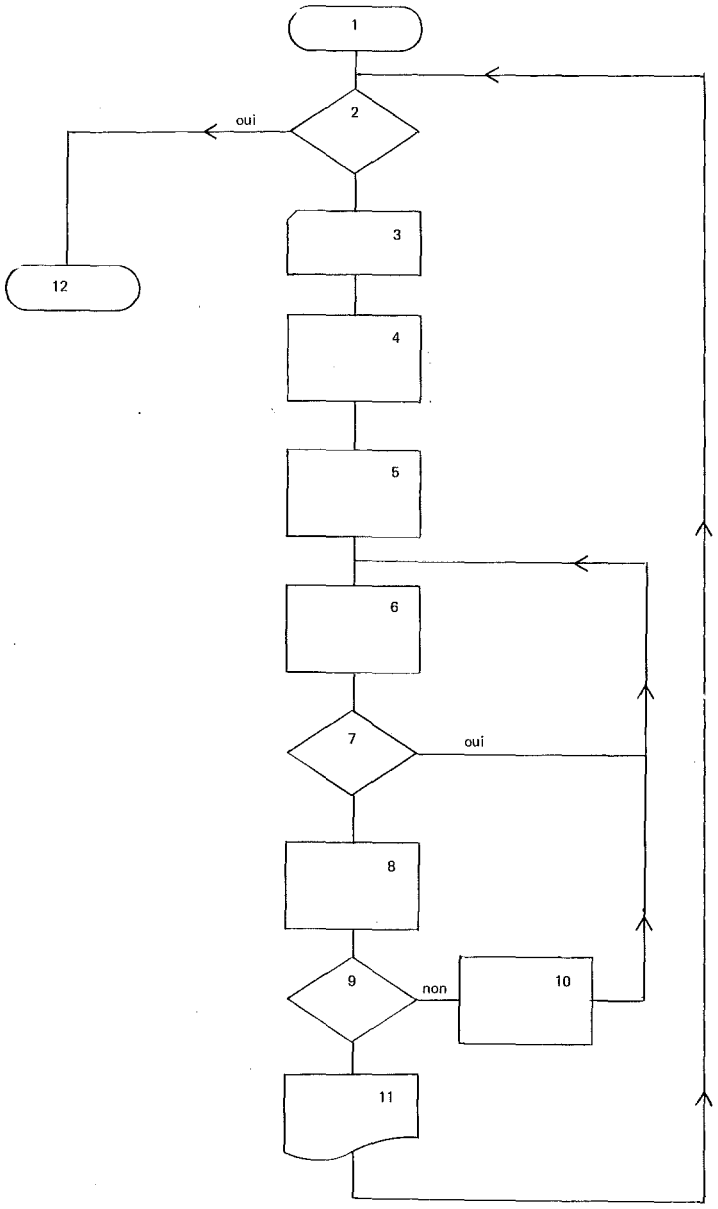
Commentaires

1) Début du programme.
2) Test de fin de fichier. Après le traitement de la dernière carte, l'ordinateur saute à l'ordre 13.
3) Lecture d'une carte dans la zone ZA.
4) Initialisation de L à 0. Cet indice sert à parcourir ZA position par position.
5) Addition de 1 à la valeur de L.
6) Interrogation sur le contenu de la position L de la zone ZA. Si cette position est blanche -c'est-à-dire si on est arrivé à la fin du mot - l'ordinateur passe à l'instruction 7, autrement il remonte à la figure 5.
7) Initialisation à 0 de l'indice J que l'on utilise pour parcourir la zone ZB.
8) Addition de 1 à la valeur contenue dans J. Lorsqu'on vient de la figure 7, l'ordre 8 donne à J la valeur 1 qui désigne ainsi la première position de ZB.
9) Transmission dans la position J de ZB de la position L de ZA. La dernière lettre du mot lu dans ZA va dans la position 1 de ZB.
10) Interrogation sur la valeur de L. Si cet indice vaut 1, l'ordinateur passe à l'instruction 12. L'inversion du mot est terminée.
11) Soustraction de 1 à la valeur de l'indice L. Cette opération permet de remonter position par position vers le début du mot.
12) Impression du résultat. Le contenu de la zone ZB est imprimé.
13) Fin du programme.

Solution 2

Le traitement se déroule en une seule phase. La zone de lecture *ZA* est parcourue de la droite vers la gauche grâce à un indice L initialisé à 21. Aussi longtemps que cette zone contient des blancs, l'ordinateur soustrait 1 à la valeur de l'indice L et continue à parcourir la zone *ZA*. Dès qu'il rencontre une lettre, l'ordinateur la transmet dans la zone *ZB* en utilisant l'indice J pour désigner successivement chaque position de *ZB*.

L'ordinogramme



Commentaires

1) Début du programme.
2) Test de fin de fichier. Après le traitement de la dernière carte, l'ordinateur exécute l'ordre 12.
3) Lecture d'une carte dans ZA.
4) Initialisation de L à 21.
5) Initialisation de J à 1.
6) Soustraction de 1 à la valeur de L.
7) Interrogation sur le contenu de la position L de ZA. Si celle-ci est blanche, l'ordinateur remonte immédiatement à l'instruction 6; autrement il passe à l'ordre 8.
8) Transmission dans la position J de ZB de la position L de ZA. Lorsque cet ordre s'exécute pour la première fois, l'indice J vaut 1 - c'est la valeur initiale qu'il a reçue en 5 - tandis que L a déjà été modifié et a une valeur qui correspond à la position de la dernière lettre du mot contenu dans la zone ZA.
9) Si L est égal à 1, le contrôle passe à l'ordre 11 puisque le traitement du mot contenu dans ZA est terminé, autrement l'ordre 10 est exécuté.
10) Addition de 1 à la valeur de l'indice J.
11) Impression du contenu de ZB. Ensuite retour à 2.
12) Fin du programme.

Dans cette solution, la zone ZA est parcourue une seule fois de droite à gauche puisque la transmission dans ZB est soumise à la condition que la position de ZA qui correspond à l'indice L ne contienne pas un blanc. Il faut noter que c'est la totalité de la zone ZA qui est parcourue alors que, dans la première solution, seule la partie de ZA qui contient des lettres est traitée.

On notera également que la solution 2 comporte une figure en moins que la solution 1.

Quelle est la meilleure solution ?

Une analyse sommaire tendrait à faire dire que c'est la seconde :
1) Elle est plus courte (une instruction en moins).
2) Elle doit être plus performante puisque la zone ZA n'est parcourue qu'une seule fois.

En ce qui concerne la longueur, la différence est négligeable, elle ne peut être un critère de choix. La rapidité du traitement, quant à elle, sera dans certains cas, supérieure dans la solution 1 bien que la zone ZA soit traitée deux fois. En voici la raison. Dans la solution 2, l'indice L varie de 21 à 1. Il y a donc, pour une même donnée, 20 répétitions des instructions 6 et 7. Pour la solution 1, il est impossible de dire *a priori* le nombre de fois que les ordres 5, 6 et 10, 11 sont exécutés. Cela dépend du nombre de lettres du mot à traiter. Ainsi, si le mot compte 6 lettres, on exécutera six fois chacune des instructions.

En fait la méthode doit être choisie en fonction du nombre moyen de lettres par mot dans la langue que l'on étudie. C'est ainsi, qu'en latin, la moyenne de lettres par mot étant de 6, l'ordinateur devra exécuter, si l'on utilise la deuxième solution pour un fichier de 10.000 mots : 10.000 mots x 20 répétitions x 2 instructions = 400.000 instructions.

Tandis que le calcul sera le suivant pour la première solution : 10.000 mots x 6 répétitions x 4 instructions = 240.000 instructions.

Il y a donc entre les deux méthodes une différence, pour un fichier relativement peu volumineux, de 160.000 instructions. Ce n'est certes pas négligeable. Toutefois il n'est pas impossible que la solution 2 convienne mieux à une langue dont le nombre moyen de lettres par mot est plus élevé.

Comme on le voit, il se confirme que, dans un problème donné, la solution à adopter est davantage une question d'équilibre et de bon sens que le fruit d'une méthodologie stricte.

Après ces quelques exercices sur la technique des ordinogrammes, nous passerons à l'étude du langage PL/1. Dans ce chapitre sur la programmation proprement dite, il nous arrivera d'établir l'ordinogramme lorsque nous aurons à faire un programme.

LE LANGAGE DE PROGRAMMATION PL/1

Avertissement

Les ouvrages de programmation sont généralement conçus comme des manuels de référence où l'on trouve une description complète du langage. Une autre conception, moins ambitieuse mais tout aussi répandue, est de décrire un langage en vue d'applications bien définies; c'est ainsi qu'ont été publiés maints ouvrages relatifs aux applications de l'Informatique aux sciences exactes ou à la gestion commerciale (voir annexe).

Les pages qui suivent ne constituent en aucune manière une description complète du langage PL/1. Leur but est de rendre accessible aux chercheurs des sciences humaines une technique avec laquelle ils sont en général peu familiarisés.

A. L'alphabet

L'apprentissage d'une langue débute normalement par l'étude de l'alphabet propre à cette langue. De même nous aborderons le PL/1 par l'examen des caractères qui le composent.

1) Les caractères alphabétiques
Il y a en PL/1, 29 caractères alphabétiques, ce sont les 26 lettres de l'alphabet, de A à Z et les trois caractères suivants :

$ dollar
# peu différent de
@ A commercial

2) Les caractères numériques : ce sont les dix chiffres de notre système numérique.
0 1 2 3 4 5 6 7 8 9.

3) Les caractères spéciaux

. point
; point-virgule
, virgule
: deux points
< inférieur à
> supérieur à
= égal

ƀ blanc : c'est le blanc typographique. On le représente conventionnellement par un b minuscule barré (ƀ)
? point d'interrogation
' apostrophe
% pour cent
(ouverture de parenthèse
) fermeture de parenthèse
* astérisque
- moins
+ plus
/ divisé
¬ non
| ou
& et
— blanc souligné

Ces caractères sont employés dans le langage PL/1, soit comme opérateurs (le signe + est l'opérateur de l'addition), soit comme séparateurs (le blanc est le séparateur le plus fréquent entre deux mots, les ponctuations sont aussi des séparateurs).

Les 29 caractères alphabétiques et les 10 caractères numériques portent le nom de caractères *alphanumériques*. Ces caractères interviennent dans la constitution des *noms* utilisés dans un programme. Celui qui rédige un programme doit y introduire des noms choisis librement pour identifier :

- le programme;
- les zones de travail (zone de lecture, compteurs, etc.);
- les noms des fichiers;
- les étiquettes, c'est-à-dire les points du programme vers lesquels l'ordinateur doit effectuer un saut.

Ces noms doivent répondre aux règles suivantes :
1) Ils doivent obligatoirement *commencer par un caractère alphabétique*; les dix chiffres peuvent être utilisés dans les noms que le programmeur choisit, mais jamais comme caractère initial du nom.
2) Les caractères spéciaux ne peuvent pas être utilisés pour constituer un nom car ils

servent de séparateurs ou d'opérateurs. Toutefois le signe que l'on appelle blanc souligné (—) peut entrer dans la composition d'un nom.
3) Les noms ont au minimum 1 caractère et au maximum 31. Ce nombre est théorique car certains ordinateurs imposent d'utiliser des noms plus courts. Dans la pratique, il me semble qu'il vaut mieux utiliser des noms qui ont au maximum six ou sept caractères.

Voici quelques exemples de noms, ceux qui sont erronés sont accompagnés d'un commentaire qui explique l'erreur.

A
ADF3
5ABCD ce nom commence par un chiffre
AB—CD
AB CD ce nom contient un blanc
AB +CD ce nom contient un opérateur (+)
ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890 ce nom est trop long.

B. La présentation d'un programme en PL/1

Lorsque l'on doit mettre par écrit un texte quelconque, la présentation matérielle peut avoir une grande importance. De même un programme écrit en PL/1 doit être présenté à l'ordinateur en respectant quelques règles.

1) Les instructions sont perforées dans les colonnes 2 à 72 des cartes mécanographiques. La colonne 1 ne peut jamais contenir de perforation. Les colonnes 73 à 80 peuvent être blanches ou peuvent contenir un numéro qui indique la place de la carte dans le programme (numéro d'ordre séquentiel).

2) Chaque carte peut contenir une ou plusieurs instructions dans les limites des colonnes 2 à 72.

3) Les mots qui composent les instructions doivent être séparés les uns des autres par un des caractères spéciaux dont j'ai donné la liste précédemment.

4) Une instruction *se termine obligatoirement par un point-virgule* (;).

C. Le début et la fin d'un programme

Un programme écrit en PL/1 commence toujours par le nom du programme et par la phrase suivante :

nom : *PROCEDURE OPTIONS (MAIN);*

- le *nom* choisi par le programmeur doit répondre aux règles que j'ai exposées précédemment.
- *les deux points qui suivent* le nom sont obligatoires; ils permettent à l'ordinateur de reconnaître *les étiquettes*. L'étiquette de la première instruction est en fait le *nom du programme*.

  PROCEDURE dont l'abréviation est *PROC* doit suivre les deux points. Ce mot annonce à l'ordinateur le début d'un programme.
- *OPTIONS (MAIN)* : ces deux mots signifient que l'on donne à l'ordinateur un *programme principal* par opposition aux sous-programmes.
- le *point-virgule* est la ponctuation obligatoire à la fin d'une instruction.

Exemples : *PROGR1 : PROCEDURE OPTIONS (MAIN);*
PROGR1 : PROCEDURE OPTION (MAIN); dans ce cas il y a une erreur car le mot *OPTIONS* doit être au pluriel; l'*S* final a été omis;
PROGR1 : PROC OPTIONS (MAIN);

La dernière instruction d'un programme est *END;*

Exemple : *END PROGR1;* cette instruction signifie "fin du programme dont le nom est *PROGR1*". D'autre part, le mot *END* peut être précédé comme toute instruction d'une étiquette et de deux points, comme dans l'exemple ci-dessous :
FIN : END;
Le mot *FIN* est une étiquette qui permettra au programme d'effectuer un saut à la dernière instruction lorsqu'un traitement sera terminé.

Entre la première instruction d'un programme et la dernière, on trouve les opérations qui constituent le programme proprement dit. Je rappelle qu'un langage se compose de deux types d'instructions, les instructions non-exécutables que l'on appelle les déclarations et les instructions exécutables, c'est-à-dire les opérations. Je commencerai par décrire les déclarations.

LES DECLARATIONS : GENERALITES

En PL/1, les déclarations de fichiers et les déclarations de zones débutent toutes par le mot *DECLARE* ou par son abréviation *DCL*. Ce mot doit être suivi du nom de l'élément que l'on définit; ce nom, choisi par le programmeur, doit respecter les règles de composition des noms que j'ai citées précédemment. Si l'on excepte ces deux points communs, les déclarations de zones et les déclarations de fichiers diffèrent totalement.

A. Les déclarations de fichier

Dans ce type de définition, on trouve à la suite du mot *DECLARE* et du nom de l'élément déclaré, le mot *FILE* qui signifie à l'ordinateur que l'on définit un fichier. On a donc au moins les trois mots suivants : *DECLARE FICHIER1 FILE* ..... ou *DCL FICHIER1 FILE*.

Après ces mots, on donne des précisions sur le type de fichier que l'on veut utiliser. Deux indications sont généralement nécessaires :

1) Soit le mot *INPUT*, soit le mot *OUTPUT* qui définissent respectivement un fichier d'entrée ou un fichier de sortie. Un fichier déclaré *INPUT* est destiné à percevoir des informations et sera associé à une opération de lecture. Un fichier défini *OUTPUT* permettra à l'ordinateur de communiquer des résultats par l'intermédiaire d'une instruction d'écriture.

2) Soit le mot *RECORD*, soit le mot *STREAM*. L'un indique que les données seront traitées une par une, l'autre s'applique à une lecture en continu. En ce qui concerne les applications de sciences humaines, on déclarera le plus souvent des fichiers du type *RECORD*.

Les précisions qui précèdent constituent les éléments de base des déclarations de fichiers. Nous reviendrons dans la suite à ces définitions, mais ce que nous venons d'en voir suffit pour faire quelques exercices.

Voici quelques exemples de déclarations de fichiers.

1.- *DECLARE FICH FILE RECORD INPUT;*
DCL FICH FILE INPUT RECORD;

Ces deux déclarations sont équivalentes, elles définissent un fichier nommé *FICH* qui sera un fichier d'entrée dans lequel les données seront lues une par une (*RECORD*).

On observera deux différences entre ces déclarations; elles sont sans importance pour l'ordinateur, en effet :
a) on peut utiliser soit le mot *DECLARE*, soit son abréviation *DCL*;
b) à la suite du mot *FILE*, l'ordre des éléments est indifférent, on peut écrire *RECORD INPUT* ou *INPUT RECORD*.

2.- *DCL FICH2 FILE RECORD OUTPUT;*

Le fichier *FICH2* est un fichier sortie (*OUTPUT*); les données sont traitées une par une.

B. Les déclarations de zones

Il faut distinguer deux types de zones, les zones alphanumériques et les zones numériques.

Les premières sont destinées à stocker les lettres de l'alphabet, les signes spéciaux et les chiffres sur lesquels on ne doit pas effectuer d'opérations arithmétiques. C'est dans ce type de zones que l'on enregistrera notamment des informations linguistiques.

Les secondes peuvent recevoir uniquement des chiffres. Elles servent aux opérations de calcul. En fait ce sont les compteurs.

Une zone occupe dans l'unité centrale de traitement un certain nombre d'octets en fonction de la longueur de la donnée que l'on veut enregistrer. Le nombre d'octets doit figurer dans la déclaration de la zone.

Le type de zones et la longueur sont des éléments que l'on trouve dans les définitions à la suite du nom choisi par le programmeur. Ces éléments s'appellent les *attributs*.

1) Les zones alphanumériques

L'attribut des zones alphanumériques est donné par le mot *CHARACTER* qui peut s'abréger en *CHAR.*

Exemple : *DCL ZL CHARACTER*
ou
DCL ZL CHAR

Ce début de déclaration annonce une zone appelée *ZL* destinée à contenir des informations alphanumériques.
Il manque à cette déclaration les indications sur la longueur de la zone. Elles seront

fournies sous la forme d'un chiffre entre parenthèses : (5) indique une zone de 5 positions dans l'unité centrale. En PL/1, une zone *CHARACTER* a au minimum 0 position et au maximum 32,767 positions.

Exemple : *DCL ZL CHAR (80);*

ZL est déclaré comme une zone alphanumérique de 80 positions. L'attribut de type précède obligatoirement l'attribut de longueur.

2) Les zones numériques

Une zone numérique se définit par quatre attributs : la base, le genre, la précision et le mode. Il n'est guère intéressant pour les recherches de sciences humaines d'examiner dans le détail tous ces attributs, je me limiterai à l'essentiel.

a) *La base* d'une zone numérique peut être *décimale* ou *binaire*.

Un nombre en base décimale est représenté normalement par les chiffres arabes. Ainsi le nombre 123 s'écrira simplement 123 en base décimale.

En base binaire, les valeurs numériques sont représentées par des bits 0 et par des bits 1. J'ai parlé du système binaire dans le paragraphe consacré à l'unité centrale de traitement (identification de l'information). Le chiffre 4 en base binaire sera codé 0100 et cinq sera écrit 0101. En mode décimal, l'attribut est *DECIMAL* (abréviation *DEC*); en mode binaire, il est *BINARY* (abréviation *BIN*).

b) *Le genre* d'une donnée arithmétique est virgule-fixe (*FIXED-POINT*) ou virgule-flottante (*FLOATING-POINT*).

Dans une donnée en virgule-fixe, le point qui sépare la partie entière et la partie décimale d'une valeur se trouve toujours à la même place, autrement dit le nombre de chiffres avant et après la virgule est constant. Ainsi, pour représenter une valeur qui a trois chiffres avant la virgule et deux après, il faudra déclarer une zone de 5 positions avec le point décimal ou binaire avant les deux dernières positions; le nombre 123,56 sera représenté par 123.56. L'attribut d'une donnée en virgule-fixe est *FIXED*. Les données à virgule-flottante intéressent surtout ceux qui doivent effectuer des calculs avec des nombres très élevés; je n'en parlerai pas.

c) *La précision* est, en fait, la longueur de la zone numérique. Lorsque c'est nécessaire on y trouve une indication sur la position de la virgule. La précision s'indique par une

valeur placée entre parenthèses à la suite d'un des attributs que nous venons de voir.

Voici quelques exemples pour des valeurs en *FIXED DECIMAL* :

(5) : le nombre peut avoir 5 chiffres entiers.

(5,2) : le nombre aura 5 chiffres, c'est ce qu'indique le 5 avant la virgule. La virgule permet à l'ordinateur de savoir que le nombre pourra comporter des chiffres décimaux; le 2 précise le nombre de chiffres décimaux. L'ordinateur sait donc qu'il doit réserver une zone pour des nombres de 5 chiffres avec deux positions décimales; en soustrayant 2 à 5, il trouve que le nombre pourra avoir au maximum 3 chiffres avant la virgule.

d) *Le mode* concerne plus particulièrement les sciences exactes, il est relatif aux *nombres réels* et aux *nombres complexes*; il est inutile d'en parler.

e) *Exemple de déclarations FIXED DECIMAL*

Les nombre décimaux à virgule-fixe sont caractérisés comme nous l'avons vu par la base DECIMAL et le genre *FIXED*. La précision minimum d'un nombre en virgule-fixe est 1 et sa précision maximum est 15. On ne peut donc jamais traiter en *FIXED DECIMAL* un nombre qui a plus de 15 chiffres que ceux-ci soient entiers ou décimaux. Enfin, un chiffre déclaré *FIXED DECIMAL* occupe un demi octet dans l'unité centrale.

DECLARE X FIXED DECIMAL (7); X sera une zone décimale à virgule-fixe dans laquelle on rangera 7 chiffres. Il n'y aura pas de chiffres après la virgule.

DECLARE X DECIMAL FIXED (7,0); cette déclaration équivaut à la précédente puisque le 0 après la virgule signifie qu'il n'y aura pas de chiffre décimal.

DECLARE X DECIMAL FIXED (9,2); X sera une zone de 9 positions numériques; elle aura 7 chiffres avant le point-décimal et 2 après.

DECLARE X FIXED DEC (17); cette déclaration est fausse; la précision maximum d'une zone *FIXED DECIMAL* est 15.

f) Exemples de déclarations FIXED BINARY

Les nombres binaires à virgule-fixe se définissent par les attributs *BINARY* pour la base et *FIXED* pour le genre. La précision d'un nombre en valeur binaire s'exprime en nombre de bits. Le nombre maximum de bits est 31, c'est-à-dire 4 octets.

DECLARE I FIXED BINARY (15); cette déclaration réserve dans l'unité centrale une

zone de 15 bits (2 octets) qui peut contenir une valeur égale à 2^{15} soit 32768.

DECLARE J FIXED BINARY (45); on ne peut déclarer une zone binaire de 45 bits; la longueur maximum admise est de 31 bits.

g) Les zones PICTURE

Les zones *PICTURE* sont des zones dites à *caractères numériques* qui offrent de grandes facilités pour la saisie des informations numériques et pour la sortie des résultats. L'attribut *PICTURE* que l'on abrège en *PIC* peut être considéré comme la base de la zone, il remplace donc dans une déclaration les mots *DECIMAL* ou *BINARY*. Un chiffre traité en *PICTURE* occupe un octet dans l'unité centrale.

La précision et le genre d'une zone *PICTURE* s'expriment par une série de caractères qui donnent l'image de la zone que l'on veut définir. En voici des exemples :

DECLARE X PICTURE '99999'; cette instruction définit une zone numérique de 5 positions. On observe grâce à cet exemple que :
1) Le chiffre 9 permet de définir une position numérique;
2) L'image de la zone composée d'un ou de plusieurs chiffres 9 est placée *entre apostrophes.*

DECLARE X PICTURE '(5)9'; cette déclaration est identique à la précédente. Le chiffre 9 est précédé d'un chiffre qui indique le nombre de positions numériques que l'on désire réserver. Ce chiffre, placé entre parenthèses après l'ouverture d'apostrophes, est un *facteur de répétition.*

DECLARE X PIC '999V99'; la zone *X* a dans ce cas 5 positions, trois avant la virgule et deux après. La lettre *V* indique la position du point décimal.

DECLARE X PICTURE '(5)9V(2)9'; cette graphie plus complexe montre que le facteur de répétition peut être employé avant et après la lettre *V* pour indiquer la répétition des chiffres de la partie entière et de la partie décimale du nombre. L'expression (5)9 définit un nombre de cinq chiffres entiers, la lettre *V* donne la place du point décimal et le reste de l'image (2)9 spécifie qu'il y aura deux décimales possibles dans le nombre.

Le PL/1 compte donc trois types de zones numériques, les zones *DECIMAL, BINARY* et *PICTURE*. Quel type faut-il choisir dans un problème déterminé ? Il est difficile de répondre à cette question; c'est la nature des données qui fait choisir tel type de

zones plutôt que tel autre; nous aurons l'occasion d'y revenir au cours des exercices que nous ferons dans la suite.

Toutefois, il faut noter que pour la lecture de données numériques on choisira en général une zone *PICTURE*. Il en ira de même pour l'écriture de résultats avec des fichiers *RECORD*.

3) La valeur initiale des zones alphanumériques et numériques

Lorsque l'on définit une zone dans un programme, l'ordinateur se contente de lui attribuer dans l'unité centrale le nombre d'octets qu'elle requiert sans se soucier des valeurs contenues dans ces octets. Ainsi une zone peut contenir des informations qui viennent d'un programme précédent et qui risquent de fausser complètement les résultats si le programmeur a omis de prendre certaines précautions. La meilleure façon d'éviter de tels incidents est d'attribuer *une valeur initiale* à une zone au moment où on la définit. Cela peut se faire en utilisant le mot *INITIAL* dont l'abréviation est *INIT*.

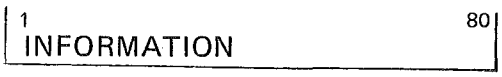
a) Pour les zones alphanumériques, la valeur initiale s'indique entre deux apostrophes, le tout étant placé entre parenthèses.

Exemples :

DECLARE A CHARACTER (11) INITIAL ('information'); le mot *information* est la valeur que l'on attribue à la zone *A*.

DECLARE A CHARACTER (11) INIT ('b'); la valeur de *A* au départ du programme sera une suite de 11 positions blanches. Le blanc typographique entre les deux apostrophes est répété automatiquement par l'ordinateur autant de fois qu'il y a de positions dans *A*.

DECLARE A CHARACTER (80) INIT ('information'); la zone *A* occupe 80 positions dans l'unité centrale; la valeur initiale qu'on lui donne est *information*. Or ce mot qui n'a que onze lettres va se placer au début de la zone *A*, dans les positions 1 à 11. L'ordinateur va placer automatiquement dans les positions 12 à 80 des blancs de sorte que la zone *A* se présentera comme suit :



b) Pour les zones numériques, la valeur initiale se place entre parenthèses. Il n'y a pas d'apostrophes.

Exemples :

DECLARE X PIC '99999' INIT (15432); la valeur 15432 est placée dans le compteur *X* au départ du programme.

DECLARE X PICTURE '(5)9' INITIAL (0); l'ordinateur met le chiffre 0 dans les cinq positions de *X*. Le compteur est donc initialisé à 0.

DECLARE X FIXED DECIMAL (5) INIT (0); ici encore, le compteur *X* est initialisé à 0.

DECLARE X FIXED DEC (5,2) INIT (234.56); la zone *X* qui comporte trois positions entières et deux décimales est initialisée à la valeur 234,56.

DECLARE X FIXED DEC (5,2) INIT (5.6); la zone *X* reçoit comme valeur initiale 5,6; les deux premières positions de *X* sont mises à 0 et la dernière (la deuxième après le point décimal) est également mise à 0.

Les pages qui précèdent donnent une description des éléments fondamentaux qui interviennent dans la déclaration des zones les plus élémentaires que l'on peut utiliser en PL/1. Il nous faudra revenir dans la suite à d'autres types de déclarations mais auparavant nous étudierons quelques instructions exécutables afin de pouvoir faire aussitôt que possible des exercices.

LES INSTRUCTIONS EXECUTABLES

1. Les instructions d'entrée et de sortie.

La lecture d'une information se fait par l'intermédiaire de l'instruction suivante :

READ FILE (nom du fichier) *INTO* (nom de la zone de lecture); ce qui signifie :

LIRE LE FICHIER (nom du fichier) *DANS LA ZONE* (nom de la zone);

Le nom du fichier et le nom de la zone doivent avoir été déclarés au préalable. Le fichier doit être un fichier d'*INPUT* puisque l'opération de lecture est une opération d'entrée de données.

Exemple :

```
DCL FICH FILE RECORD INPUT;
DCL ZL CHAR (80);
READ FILE (FICH) INTO (ZL);
```

Le fichier *FICH* a été déclaré *INPUT*, il peut donc être utilisé dans une opération de lecture.

ZL est une zone de 80 positions alphanumériques que l'on va employer comme zone de lecture.

L'opération *READ FILE (FICH) INTO (ZL);* amène une donnée dans la zone *ZL* située dans l'unité centrale. Le fichier se trouve sur cartes mécanographiques, l'opération de *READ* amènera dans la zone *ZL* le contenu d'une carte.

L'écriture d'un résultat sur un support externe se fait par l'opération :

WRITE FILE (nom du fichier) *FROM* (nom de la zone d'écriture); ce qui signifie :

ECRIRE SUR LE FICHIER (nom du fichier) *A PARTIR DE* (nom de la zone);

Le fichier doit avoir été déclaré au préalable avec l'attribut *OUTPUT*.

Exemple :

```
DCL FICH2 FILE RECORD OUTPUT;
DCL ZECR CHAR (75);
WRITE FILE (FICH2) FROM (ZECR);
```

Avec les deux instructions que nous venons de voir, nous pouvons écrire un programme qui consiste à *lire un fichier de cartes mécanographiques* et à *imprimer le contenu de ces cartes*. Ce programme est en fait une version simplifiée de l'exercice n°1 du chapitre consacré aux ordinogrammes.

```
1 EX1 : PROC OPTIONS(MAIN);
2 DCL FIC FILE RECORD INPUT;
3 DCL FIC1 FILE RECORD OUTPUT;
4 DCL ZL CHAR(80);
5 READ FILE(FIC) INTO(ZL);
6 WRITE FILE(FIC1) FROM(ZL);
7 END;
```

La première et la dernière lignes du programme ne demandent aucun commentaire; on y trouve les instructions de début et de fin de programme. Les instructions 2, 3 et 4 sont les déclarations des fichiers d'entrée et de sortie et de la zone de lecture-écriture. Ces instructions appellent les remarques suivantes :

a) L'ordre de présentation des déclarations est indifférent. On aurait pu définir la zone *ZL* avant les fichiers.

b) La zone *ZL* sert à la fois de zone de lecture et de zone d'écriture. Puisque le programme consiste simplement à lire des données et à les imprimer sans leur faire subir de modifications, on peut imprimer à partir de la zone de lecture.

c) La zone *ZL* n'est pas initialisée : au départ du programme, elle peut contenir n'importe quoi y compris des informations venant d'un autre programme. Le contenu de *ZL* n'a aucune importance à ce moment car l'opération de lecture va amener dans *ZL* une donnée correcte et effacer simultanément toutes les informations qui auraient pu s'y trouver.

La séquence des instructions 5 et 6 ne peut être modifiée. La lecture doit se faire avant l'écriture.

2. Le saut inconditionnel.

L'ordinateur exécute un programme d'une manière séquentielle et répétitive. Dans le programme ci-dessus, la séquence d'instructions exécutables commence à l'opération 5, c'est-à-dire à la lecture du fichier *FIC*. Après quoi l'ordinateur exécute l'ordre 6 et l'ordre 7. Cela signifie que l'ordinateur va lire *une donnée*, l'imprimer puis il s'arrêtera. Dès lors, il n'y a pas répétitivité du processus de lecture-écriture. Il manque par conséquent un ordre de retour à l'opération 5, c'est-à-dire *un saut inconditionnel*. Le saut inconditionnel se fait par l'instruction suivante :

GOTO étiquette;
ou
GO TO étiquette;

L'étiquette est un nom choisi par le programmeur; elle doit figurer au moins en deux endroits du programme. D'une part, après le mot *GOTO*, dans une instruction de saut, et, d'autre part, avant l'instruction à laquelle l'ordinateur doit sauter. Devant une instruction, l'étiquette doit être suivie de deux points.

Exemple :

DEBUT : READ FILE(FIC) INTO(ZL);
.....
.....
GOTO DEBUT;

Le mot *DEBUT* est l'étiquette de l'instruction *READ FILE...*, il est séparé de celle-ci par deux points. A un endroit déterminé du programme, l'ordinateur doit remonter à l'opération de lecture *(READ FILE...*; on utilise l'instruction *GOTO* suivie de l'étiquette qui identifie l'opération que l'on veut exécuter.

L'instruction de saut inconditionnel permet donc d'assurer la répétitivité d'un programme.

Le programme *EX1* peut être complété :

.....
.....
5 DEBUT : READ FILE(FIC) INTO(ZL);
6 WRITE FILE (FIC1) FROM(ZL);
7 GOTO DEBUT;
8 END;

3. Le test de fin de fichier.

Un dernier problème subsiste. L'instruction *GOTO* va permettre de répéter les ordres de lecture et d'écriture autant de fois que l'on veut. Or, ce que l'on veut, c'est que l'ordinateur s'arrête lorsqu'il aura lu toutes les cartes du fichier *FIC*. Dès lors, il faut ajouter un ordre conditionnel d'arrêt du programme après le traitement de la dernière carte. Cet ordre sera : *ON ENDFILE* (nom du fichier d'*INPUT*) *GOTO* étiquette;

ce qui signifie :

A LA FIN DU FICHIER (nom) *ALLER A* étiquette;
Cette instruction doit se placer dans la partie du programme réservée aux déclarations.

Le nom du fichier entre parenthèses doit être le fichier d'*INPUT* dont on veut repérer la fin.

Reprenons l'exercice 1 et complétons-le :

```
1 EX1 : PROC OPTIONS(MAIN);
2 DCL FICH FILE RECORD INPUT;
3 DCL FICH1 FILE RECORD OUTPUT;
4 DCL ZL CHAR(80);
5 ON ENDFILE(FICH) GOTO FIN;
6 DEBUT : READ FILE(FICH) INTO(ZL);
7         WRITE FILE(FICH1) FROM(ZL);
8         GOTO DEBUT;
9 FIN : END;
```

Le test de fin de fichier en PL/1 est mis en place au début du programme; à chaque ordre de lecture, l'ordinateur contrôle si le fichier est terminé. Aussi longtemps qu'il reste des données à lire la séquence 6, 7 et 8 est exécutée. A la fin du fichier *FICH*, l'ordinateur saute à l'étiquette *FIN* où il rencontre l'instruction d'arrêt *END*.

4. Les opérations arithmétiques.

L'ordinogramme n° 1 comporte en plus des ordres de lecture et d'écriture, une instruction qui compte les données et une autre qui imprime le nombre de cartes traitées. Pour réaliser le programme qui correspond à cet ordinogramme, il est nécessaire de connaître les instructions de calcul.

En PL/1, la présentation des quatre opérations fondamentales est très proche des expressions algébriques. L'utilisation des instructions de calcul requiert l'emploi de compteurs; les voici :

```
DECLARE X FIXED DECIMAL(10);
DECLARE Y DEC FIXED(5);
DECLARE Z DEC(5) FIXED;
```

On décide d'attribuer à *Y* la valeur 10 et à *Z* la valeur 5.

a) L'addition

```
X = Y + Z;
```

Cette opération provoque la totalisation des valeurs contenues dans *Y* et dans *Z*; le résultat est transféré dans le compteur *X* qui vaut alors 15. Il faut remarquer que le contenu

de *Y* et celui de *Z* restent inchangés.

X = X + 1;

En exécutant cet ordre, l'ordinateur ajoute la valeur 1 au nombre contenu dans *X*. Si au départ *X* vaut 0, l'instruction *X = X + 1*, lui donne la valeur 1.

b) Soustraction

X = Y - Z;
X = X - 1;

Dans le premier cas, l'ordinateur soustrait la valeur de *Z* de la valeur de *Y* et il transmet le résultat dans la zone *X* qui vaut à ce moment 5. Le contenu de *Y* et celui de *Z* restent inchangés. La seconde expression soustrait 1 de la valeur de *X*. Si *X* est égal à zéro, l'instruction *X = X - 1* donnera à *X* la valeur -1.

c) La multiplication

L'opérateur de la multiplication est le signe *.

*X = Y * Z;*
*X = X * 2;*

La première opération consiste à multiplier la valeur de *Y* par celle de *Z*; le résultat, qui dans ce cas est 50, est transmis dans la zone numérique *X*. La deuxième expression provoque la multiplication par 2 du contenu de *X*.

d) La division

L'opérateur de la division est le signe /.

X = Y / Z;
X = Y / 2;

L'expression *X = Y / Z* est la division de la valeur de *Y* par celle de *Z*; le résultat de cette division qui est 2 est transféré dans la zone *X*. La deuxième opération consiste à diviser la valeur de *Y* par 2 et à mettre le résultat dans *X*.

5. L'instruction PUT EDIT.

Nous possédons maintenant toutes les informations pour écrire le programme qui sert à lire des cartes, à les écrire et à les compter puis, en fin de fichier, à imprimer le nombre de données traitées. Cependant avant de rédiger ce programme, je voudrais parler

d'une instruction d'impression qui présente de nombreuses facilités d'utilisation, *l'instruction PUT EDIT*. Cet ordre d'écriture provoque la *déclaration automatique* d'un fichier de sortie sur imprimante; au surplus, il permet de réaliser des mises en page complexes des données. Voici la présentation générale du *PUT EDIT*.

PUT EDIT (nom de la zone à imprimer)(contrôle de la présentation des documents ainsi que type et longueur de la zone à imprimer);

Le nom de la zone à imprimer se trouve dans un premier groupe de parenthèses. S'il y a plusieurs zones, on sépare les noms les uns des autres par une virgule. Par exemple, pour imprimer *X*, *Y* et *Z*, on écrira *PUT EDIT (X, Y, Z) (.....);*

La deuxième série de parenthèses est destinée à recevoir les indications que l'on appelle les *formats* d'impression.

Il s'agit d'une part, d'informations qui guident le travail de l'imprimante dans la mise en page; elles concernent les sauts à la page et à la ligne, la position dans la ligne où l'on veut commencer l'impression (justification à gauche), etc. Parmi ces indications, je citerai :

a) *PAGE* : ce mot provoque un saut du papier à une nouvelle page.
b) *SKIP* : cette indication entraîne un saut à la ligne.
   SKIP(n) : au lieu de passer simplement une ligne, l'ordinateur en saute n. Si n vaut 3, l'ordinateur laisse trois lignes blanches. On peut avoir à la suite du mot *SKIP*, entre parenthèses, soit un chiffre soit un nom de zone numérique. Dans ce cas, l'ordinateur passe un nombre de lignes équivalent à la valeur de la zone.
c) *COLUMN(m)* : ce mot indique le numéro de la position dans la ligne où l'on veut commencer l'impression (justification à gauche); m peut être un chiffre ou un nom de compteur.

Outre ces informations de mise en page, on trouve également dans le second groupe de parenthèses, des indications concernant le type (alphanumérique, décimal, binaire) et la longueur des zones à imprimer. Le type est symbolisé par une lettre et la longueur s'indique entre parenthèses par une valeur qui correspond à celle qui a été donnée lors de la déclaration de la zone. Les zones alphanumériques sont représentées par la lettre *A* et les zones alphanumériques par la lettre *F*. Ainsi, *A(80)* représente une zone alphanumérique de 80 positions et *F(5)*, un compteur de 5 positions.

Exemple :

On veut imprimer sur une même ligne deux informations, la première est stockée dans une zone alphanumérique de 60 positions qui s'appelle *ZE*; la deuxième est un compteur (*Y*) de 5 positions. L'instruction d'écriture sera :

PUT EDIT(ZE,Y) (A(60),F(5));

Si l'on veut des blancs typographiques entre la zone *ZE* et la zone *Y*, on utilisera un format d'espacement qui est symbolisé par la lettre *X*. Cette lettre est suivie d'une valeur entre parenthèses qui précise le nombre de blancs à laisser entre deux zones.

Par conséquent, si l'on désire insérer deux positions blanches entre *ZE* et *Y*, on devra prévoir un format *X(2)* dans la deuxième partie du *PUT EDIT*; l'instruction se présentera comme suit :

PUT EDIT(ZE,Y) (A(60),X(2),F(5));

Pour imprimer les zones *ZE* et *Y* en commençant à une nouvelle page, on écrira :

PUT EDIT (ZE,Y) (PAGE,A(60),X(2),F(5));

Si l'on veut sauter trois lignes avant d'écrire sur l'imprimante les zones *ZE* et *Y*, on utilisera le mot *SKIP* suivi du chiffre 3 entre parenthèses :

PUT EDIT(ZE,Y) (SKIP(3),A(60),X(2),F(5));

Enfin, pour laisser, avant d'imprimer les deux zones, une marge à gauche de 10 positions, on écrira *COLUMN(11)* :

PUT EDIT (ZE,Y) (COLUMN(11),A(60),X(2),F(5));

Il faut noter qu'il n'y a pas d'incompatibilité entre les mots *PAGE, SKIP* et *COLUMN*, une instruction telle que la suivante est correcte :

PUT EDIT(ZE,Y) (PAGE,SKIP(3),COLUMN(11),A(60),X(2),F(5));

Ce qui signifie : Imprimer les zones *ZE* et *Y* après avoir fait un saut à la page, passé trois lignes et en commençant à la position 11, c'est-à-dire en laissant 10 blancs comme marge à gauche.

Il resterait beaucoup de précisions à donner au sujet de l'instruction *PUT EDIT;* je pense toutefois que les indications qui précèdent suffisent à montrer la souplesse du *PUT EDIT;* elles permettent en tous cas de l'utiliser. C'est ce que nous allons voir dans l'exercice

qui suit et dont nous connaissons déjà l'énoncé.

Transposition en PL/1 de l'exercice n°1 (cf chapitre des ordinogrammes).

```
1)  EX01 : PROCEDURE OPTIONS(MAIN);
2)  DCL FICHIER FILE RECORD INPUT;
3)  DCL ZL CHAR(80);
4)  DCL X FIXED DEC(5) INIT(0);
5)  ON ENDFILE(FICHIER) GOTO FIN;
6)  DEBUT : READ FILE(FICHIER) INTO(ZL);
7)          PUT EDIT(ZL) (SKIP, A(80));
8)          X = X + 1;
9)          GOTO DEBUT;
10) FIN : PUT EDIT(X) (PAGE,F(5));
11)       END;
```

Ce programme qui est une transposition en PL/1 de l'ordinogramme n°1 appelle quelques remarques.

a) Il n'est pas nécessaire d'initialiser la zone *ZL* car l'opération *READ FILE(FICHIER) INTO(ZL);* a pour conséquence de remplacer le contenu de *ZL* quel qu'il soit par le contenu de la carte lue. Le compteur *X*, par contre, doit être initialisé; en effet, la première opération où *X* intervient consiste à ajouter 1 à sa valeur. Si cette valeur n'est pas connue et contrôlée au départ du programme, le dénombrement effectué dans *X* risque d'être erroné.

b) La séquence des instructions respecte l'ordre des figures de l'ordinogramme. On pourrait cependant modifier cet ordre. L'instruction *X = X + 1;* pourrait être placée entre l'ordre de lecture et l'ordre d'écriture.

c) L'instruction *PUT EDIT* dispense le programmeur de déclarer explicitement un fichier d'*OUTPUT* sur imprimante. L'ordinateur en rencontrant cette instruction d'écriture déclare automatiquement le fichier de sortie.

6. L'instruction de comparaison et le saut conditionnel.

Les quatre ordinogrammes que j'ai proposés pour le problème n°2 vont être transcrits en PL/1. Le lecteur en profitera pour mémoriser les instructions PL/1 déjà rencontrées et pour apprendre à les utiliser commes des formules toutes faites du langage.

Par ailleurs, nous devrons étudier d'autres instructions dont nous aurons besoin dans les programmes qui suivent : nous le ferons progressivement.

Dans les quatre solutions, nous rencontrerons une instruction conditionnelle de comparaison. En voici le schéma simplifié.

IF condition *THEN* instruction;
ELSE instruction;
ce qui signifie :

Si telle condition est remplie *ALORS EXECUTER* instruction;
AUTREMENT EXECUTER instruction;

On remarque tout d'abord que l'ordre conditionnel *IF* s'accompagne de deux instructions au moins : il y a un point-virgule après la partie *THEN* de l'ordre et un autre à la fin de la partie *ELSE*.

La condition à la suite du mot *IF* est une comparaison entre deux termes selon les rapports d'égalité ou d'inégalité, de supériorité ou d'infériorité. Nous avons parlé précédemment des huit relations logiques possibles entre deux termes.

L'instruction à la suite de *THEN* et de *ELSE* peut être n'importe quelle opération du langage PL/1 *y compris une autre condition.*

Si on imagine une zone numérique d'une position dont on veut savoir si elle vaut 2, on aura les instructions suivantes :

```
DCL X PIC '9';
```

X, Y et *W* sont des compteurs quelconques déclarés dans le programme.

```
.....
      1          2            3          4
IF X=2   THEN Y=X+2;    ELSE Z=X/2;   W=Y+Z;
```

L'ordinateur exécute le premier membre de l'opération, il vérifie si *X* vaut 2. Il estimera alors que le résultat de la comparaison est *VRAI* si *X* vaut 2 et qu'il est *FAUX* si *X* a une valeur différente de 2. Si le résultat est *VRAI*, l'ordinateur exécute la deuxième partie de l'opération (*Y = X + 2*) et saute ensuite à l'ordre qui suit la condition (*W = Y + Z;*) dans la séquence du programme. La troisième partie de l'opération *IF* est ignorée.

Par contre, si le résultat de l'évaluation *X = 2* est *FAUX*, l'ordinateur va de la première partie de l'instruction à la troisième (*ELSE*) pour exécuter *Z = X / 2;* ensuite il passe à l'ordre 4.

Il existe une forme plus simple de l'instruction *IF* dans laquelle le membre *ELSE...;* est omis si on n'en a pas besoin. Cela revient à dire :

```
    1          2          4
IF X = 2 THEN Y = X + 2; W = Y + Z;
```

Dans le cas où la troisième partie du *IF* n'existe pas, l'ordinateur exécute la séquence 1, 2, 4 si la comparaison de *X* à la valeur 2 donne un résultat *VRAI*; il saute la deuxième partie du *IF* et passe directement de 1 à 4 si le résultat est *FAUX*.

Enfin, il est important de noter qu'on ne peut avoir qu'une seule instruction à la suite de *THEN* ou de *ELSE*. L'instruction suivante est fausse car il y a deux instructions entre *THEN* et *ELSE :*

```
IF X = 2 THEN X = Y + 2; W = Y + Z; ELSE Z = X / 2;
```

L'instruction conditionnelle de comparaison dont nous aurons besoin dans les quatre programmes a pour but d'examiner la première position d'une zone qui en compte 80.

Comment peut-on en PL/1, accéder à une seule position ?

Il y a pour cela différents procédés dont j'aurai l'occasion de parler. Je retiendrai pour l'instant une méthode qui consiste à isoler au moyen d'une déclaration la position sur laquelle on doit travailler.

7. Complément aux déclarations : la structure.

Des informations qui ont des caractéristiques différentes de type et de longueur doivent quelquefois être groupées dans la même zone pour des raisons logiques. Ainsi, le nom et le prénom d'une personne, le nom de la localité et de la rue où elle habite ainsi que le numéro de la maison constituent l'adresse complète de cette personne. Ce sont tous éléments qui ont des caractéristiques propres mais qui, mis ensemble, permettent d'identifier et de retrouver quelqu'un.

Le langage PL/1 permet de regrouper dans une déclaration des informations qui ont entre elles des liens logiques; ce type de déclaration s'appelle une *structure*. Une structure est une *collection hiérarchisée* de zones. Voici quelques éléments à réunir

dans une structure :

```
NOM CHAR(25)
PRENOM CHAR(10)
LOCALITE CHAR(25)
RUE CHAR(15)
NUMERO PICTURE'(5)9'
```

On peut grouper ces éléments dans une zone structurée qui s'appellera, par exemple, *IDENTITE*. Pour cela on écrira ce qui suit :

```
DECLARE 1 IDENTITE,
        2 NOM CHARACTER(25),
        2 PRENOM CHARACTER(10),
        2 LOCALITE CHAR(25),
        2 RUE CHAR(15),
        2 NUMERO PIC '(5)9';
```

Le mot *DECLARE* est employé une seule fois devant le nom général. On a attribué à chaque nom intervenant dans la définition de la structure un numéro qui correspond à son niveau; le niveau général porte le numéro le moins élevé (ce sera toujours le niveau 1), ce niveau ne porte aucune indication de type ou de longueur, il est précisé par les éléments qui le composent.

Chaque élément de la déclaration est séparé du précédent par une virgule, seul le dernier élément est suivi d'un point-virgule. Enfin, le numéro doit être suivi d'un blanc.

On peut vouloir grouper différemment les informations contenues dans la structure *IDENTITE*, en construisant une structure à 3 niveaux. Le premier sera le nom général (*IDENTITE*); le deuxième niveau réunira, d'une part, le nom et le prénom, il s'appellera, par exemple, *INDIVIDU*, et, d'autre part, la localité, la rue et le numéro, il se nommera *ADRESSE*. La structure se présentera de la manière suivante :

```
DECLARE  1  IDENTITE,
           2  INDIVIDU,
             3  NOM CHAR(25),
             3  PRENOM CHAR(10),
           2  ADRESSE,
```

```
3 LOCALITE CHAR(25),
3 RUE CHAR(15),
3 NUMERO PIC'99999';
```

Lorsque dans un programme on fait référence au nom *IDENTITE*, on a accès à tous les éléments de la zone. Lorsqu'on fait référence à *INDIVIDU*, on accède simultanément à *NOM* et à *PRENOM;* enfin, en faisant référence à un élément du niveau le plus élevé (dans l'exemple qui précède le niveau 3 est le plus élevé), on peut traiter uniquement l'élément dont on a donné le nom.

Dans un ordre de *READ* ou de *WRITE, on doit toujours donner le nom général de la structure.* Les opérations de lecture et d'écriture ne peuvent se faire qu'à partir du niveau 1. Par contre, lorsqu'on utilise une instruction *PUT EDIT* pour imprimer le contenu d'une structure, il est préférable de donner le nom des éléments dont le niveau est le plus élevé.

Exercice 2, solution 1

Nous pouvons à présent écrire le programme qui correspond à la première solution de l'exercice 2. Nous utiliserons pour déclarer la zone de lecture une structure appelée *ZL* dont les éléments de niveau 2 seront *ZA* (une position numérique) et *ZB* (79 positions alphanumériques).

```
1)  EX2S01 : PROC OPTIONS (MAIN);
2)  DCL FICHIER FILE RECORD INPUT;
3)  DCL 1 ZL,
4)          2 ZA PIC '9',
5)          2 ZB CHAR(79);
6)  DCL (XA, XB, XC, XD, XE, XF) FIXED DEC(5) INIT(0);
7)  ON ENDFILE(FICHIER) GOTO FIN;
8)  AA : READ FILE(FICHIER) INTO(ZL);
9)          IF ZA ¬= 1 THEN GOTO AB;
10)         XA = XA + 1;
11)         GOTO AA;
12) AB : IF ZA ¬= 2 THEN GOTO AC;
13)         XB = XB + 1;
14)         GOTO AA;
```

```
15) AC : IF ZA ¬= 3 THEN GOTO AD;
16)      XC= XC + 1;
17)      GOTO AA;
18) AD : IF ZA ¬= 4 THEN GOTO AE;
19)      XD = XD + 1;
20)      GOTO AA;
21) AE : IF ZA ¬= 5 THEN GOTO AF;
22)      XE = XE + 1;
23)      GOTO AA;
24) AF : IF ZA ¬= 6 THEN GOTO AA;
25)      XF = XF + 1;
26)      GOTO AA;
27) FIN : PUT EDIT(XA) (SKIP, F(5));
28)       PUT EDIT(XB) (SKIP, F(5));
29)       PUT EDIT(XC) (SKIP, F(5));
30)       PUT EDIT(XD) (SKIP, F(5));
31)       PUT EDIT(XE) (SKIP, F(5));
32)       PUT EDIT(XF) (SKIP, F(5));
33)       END;
```

La ligne 6 demande un mot d'explication. Le programme *EX2S01* requiert la déclaration de 6 compteurs de type et de longueur identiques, on aurait pu écrire :

```
DECLARE XA FIXED DEC(5) INIT(0);
DECLARE XB FIXED DEC(5) INIT(0);
.....
DECLARE XF FIXED DEC(5) INIT(0);
```

Au lieu de répéter 6 fois les mêmes attributs pour les compteurs, on a procédé à une déclaration commune par la mise en facteur du type, de la longueur et de la valeur initiale des 6 compteurs. Pour cela il a suffi de placer après le mot *DECLARE* tous les noms à declarer entre parenthèses.

Au point de vue logique, le programme suit l'ordinogramme. L'ordinateur exécute les instructions 8 à 26 pour traiter le fichier; lorsque toutes les cartes ont été lues, on passe aux instructions 27 à 33 qui consistent à imprimer la valeur des compteurs et à arrêter le déroulement du programme.

Les instructions *IF* sont du type le plus simple sans utilisation du *ELSE*. On peut se demander pourquoi ces opérations sont rédigées de manière négative; en voici la raison.

J'ai dit que, à la suite du *THEN* ou du *ELSE*, on ne peut jamais avoir qu'une seule instruction. Or, lorsque la zone *ZA* contient le code que l'on recherche, l'ordinateur doit exécuter deux opérations à savoir l'addition de 1 dans le compteur correspondant et le saut inconditionnel à l'instruction de lecture. Dès lors, on ne pouvait écrire ce qui suit :

1) *IF ZA = 1 THEN XA = XA + 1;*
2) *GOTO AA;*
3) *IF .....*

Ainsi rédigé, le programme aurait été faux. L'ordinateur, en effet, aurait exécuté la première partie du *IF* (la comparaison avec le code 1); en cas d'égalité entre *ZA* et 1, il aurait effectué l'addition *XA = XA + 1* puis le saut à l'étiquette *AA*. Par contre, dans les cas où *ZA* ne serait pas égal à 1, l'ordinateur passerait de la première partie du *IF* à l'instruction de saut à *AA* sans exécuter l'addition *XA = XA + 1*. On voit que jamais l'ordinateur n'arriverait à la ligne 3 qui se trouve ci-dessus, c'est pourquoi on doit poser les questions de manière négative.

Exercice 2, solution 2

Cette solution, on s'en souvient, fait intervenir une table et un indice. Examinons d'abord comment on procède à la déclaration d'une table.

8. Complément aux déclarations : la table.

J'ai dit qu'une table était une zone composée d'un certain nombre d'éléments de type et de longueur identiques. La définition d'une table en PL/1 ressemble à la définition d'une zone normale à laquelle on ajoute l'indication du nombre d'éléments de la table. Cette indication est placée entre parenthèses à la suite du nom.

DECLARE X(6) FIXED DEC(5);

X est une zone numérique *(FIXED DEC)* qui peut recevoir six nombres de 5 chiffres (5). La présence du chiffre 6 après le nom de la zone indique, en effet, à l'ordinateur que *X* est une table de 6 éléments.

DECLARE Z(10) CHAR(1);

La zone alphanumérique *Z* est une zone qui se compose de 10 éléments qui occupent

chacun une position dans l'unité centrale.

L'initialisation d'une table numérique pose peu de problèmes. Ainsi, si l'on définit une table *X* de 6 éléments numériques et que l'on veut initialiser le premier élément à 10, le deuxième à 20, le troisième à 15, le quatrième à 30, le cinquième à 25 et le sixième à 35, on écrira :

DECLARE X(6) FIXED DEC(5) INIT(10, 20, 15, 30, 25, 35);

On voit qu'il suffit de placer à l'intérieur des parenthèses qui suivent le mot *INIT*, les valeurs initiales dans l'ordre où elles doivent figurer dans la table. Entre chaque valeur initiale, on trouve une virgule.

Si le premier et le deuxième éléments doivent être initialisés à 10 et les autres aux mêmes valeurs que ci-dessus, on peut écrire la déclaration de deux manières différentes :

DECLARE X(6) FIXED DEC(5) INIT(10, 10, 15, 30, 25, 35);

C'est le même procédé que dans le premier exemple.

DECLARE X(6) FIXED DEC(5) INIT((2)10, 15, 30, 25, 35);

Dans ce cas, la valeur 10 qui doit être attribuée aux deux premiers éléments de la zone est précédée d'un facteur d'itération qui est indiqué entre parenthèses immédiatement avant la valeur. Ainsi, si l'on veut définir une table en initialisant tous les éléments à 0, le facteur d'itération permettra d'écrire : *DCL X(6) FIXED DEC(5) INIT((6)0);*
L'initialisation d'une table alphabétique est semblable à celle d'une table numérique sauf en ce qui concerne l'emploi du facteur d'itération.

DCL TA(10) CHAR(1) INIT('A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J');

Les valeurs initiales sont indiquées dans la parenthèse qui suit le mot *INITIAL*; elles apparaissent dans l'ordre où elles doivent figurer dans la table. Les valeurs sont placées entre apostrophes et sont séparées les unes des autres par une virgule.

DCL TA(10) CHAR(5) INIT((5)'A');

Le chiffre 5 après le mot *INIT n'est pas un facteur d'itération. C'est un facteur de répétition* qui concerne la valeur attribuée au premier élément de la table. Cet élément doit être initialisé avec le caractère *A* de la première à la cinquième position. Les éléments 2 à 10 de *TA* ne reçoivent pas de valeur initiale.

Le *facteur d'itération* permet de reproduire dans un certain nombre d'éléments la valeur attribuée à un de ces éléments. *Le facteur de répétition d'une valeur alphanumérique* permet, lui, de reproduire cette valeur un certain nombre de fois à l'intérieur d'une seule zone.

Pour initialiser tous les éléments d'une table alphanumérique à une même valeur, il faut utiliser simultanément les deux facteurs. D'abord, le facteur d'itération ensuite le facteur de répétition. En voici un exemple :

DCL TA (10) CHARACTER(5) INIT((10) (5)'A');
ce qui signifie que l'ordinateur doit donner dix fois la valeur 'AAAAA' à la table *TA*; dès lors, les 10 éléments seront initialisés.

9. Les indices.

Une deuxième notion nécessaire à la réalisation de la solution 2 est l'indice.

Nous savons qu'un indice est un compteur numérique. Dès lors toute déclaration de zone numérique peut servir d'indice. Ainsi, *DCL X FIXED DEC(5);* : la zone numérique *X* peut être utilisée comme un indice par le programmeur. Toutefois pour des raisons de rapidité d'exécution, il est souhaitable d'utiliser des indices déclarés en *FIXED BINARY*. En effet, ce type de zones convient mieux à la logique binaire de l'ordinateur.

DCL I FIXED BIN(15);

I est un indice binaire. Le PL/1 offre la possibilité de ne pas déclarer les zones numériques binaires dans le programme. L'ordinateur procède lui-même à la déclaration d'un compteur *FIXED BINARY (15)* à condition que le nom de la zone commence par une des lettres *I, J, K, L, M* ou *N* et qu'il ne soit pas déclaré par ailleurs dans le programme. Ainsi, lorsque l'ordinateur rencontre dans un programme l'instruction *J = 0;* il déclare *J* avec les attributs *FIXED BIN(15)* à condition que *J* ne soit pas définit dans le programme.

Les notions de table et d'indice que nous venons de voir nous permettent d'écrire le programme qui correspond à la deuxième solution de l'exercice 2.

```
1)  EX02S02 : PROC OPTIONS(MAIN);
2)  DCL FIC FILE RECORD INPUT;
3)  DCL 1 ZL,
4)        2 ZA PIC '9',
```

```
 5)        2 ZB CHAR(79);
 6)    DCL X(6) FIXED DEC(5) INIT((6)0);
 7)    ON ENDFILE(FIC) GOTO FIN;
 8)    DEBUT : READ FILE(FIC) INTO (ZL);
 9)              J = 0;
10)    AB : J = J + 1;
11)         IF ZA ¬= J THEN GOTO AC;
12)         X(J) = X(J) + 1;
13)         GOTO DEBUT;
14)    AC : IF J = 6 THEN GOTO DEBUT;
15)                   ELSE GOTO AB;
16)    FIN : J = 0;
17)    FINA : J = J + 1;
18)           PUT EDIT(X(J)) (SKIP,F(5));
19)           IF J ¬= 6 THEN GOTO FINA;
20)           END;
```

L'instruction 6 a pour rôle de déclarer la table *X* comme zone numérique composée de 6 éléments qui occupent chacun 5 positions dans l'unité centrale. Cette table sera utilisée à l'opération d'addition (instruction 12) et à l'ordre d'écriture des résultats (instruction 18).

On voit que l'utilisation d'une table ne pose aucun problème; il suffit de donner le nom de la table et d'indiquer entre parenthèses le numéro de l'élément sur lequel on doit travailler. Ce numéro se donne soit sous la forme d'un chiffre soit sous la forme d'un indice variable. Aux instructions 12 et 18 de l'exercice *EX02S02*, le numéro de l'élément correspond à la valeur de l'indice J. A l'opération 15, le mot *ELSE* est inutile. En effet, on aurait pu écrire :

```
14)  IF J = 6 THEN GOTO DEBUT;
15)  GOTO AB;
```

Les deux façons d'écrire l'instruction 15 sont équivalentes; cela est dû au fait que les deux parties du *IF* sont composées de sauts inconditionnels.

Si l'on avait eu à la suite du mot *THEN* une autre instruction que le *GOTO*, l'emploi du *ELSE* aurait alors été obligatoire.

IF J = 6 THEN PUT EDIT(ZA) (SKIP,A(1));
ELSE GOTO AB;

La séquence obtenue dans ce cas est la suivante : si *J* vaut 6, l'ordinateur imprime le contenu de *ZA* et passe à la suite du programme. Si *J* ne vaut pas 6, la partie *ELSE* du *IF* commande un saut à l'étiquette *AB*. L'absence du mot *ELSE* dans ce cas aurait eu pour conséquence l'exécution de l'ordre *GOTO AB* quelle que soit la valeur de *J.*

10. L'opération d'affectation.

La solution 3 de l'exercice 2 fait intervenir une instruction d'affectation, c'est-à-dire une opération qui permet de faire passer dans une zone la totalité ou une partie du contenu d'une autre zone. En PL/ 1, c'est le signe = qui est l'opérateur de l'instruction d'affectation.

Ainsi *X = Y;*
Cette opération fait passer dans la zone *X* le contenu de *Y*. En principe les zones *X* et *Y* doivent avoir les mêmes attributs, c'est-à-dire appartenir au même type et être de longueur identique. Lorsque les deux zones sont de types différents, l'ordinateur fait appel à des sous-programmes de conversion pour transcoder la donnée d'un type à l'autre. Cette opération peut présenter certains risques. Dès lors, il est préférable d'éviter des affectations entre zones de types différents.

En ce qui concerne les différences de longueur entre zones, il y a lieu de distinguer très nettement l'affectation des zones alphanumériques et l'affectation des zones numériques.

a) Les zones alphanumériques

DECLARE A CHAR(50);
DECLARE B CHARACTER(80);

Les zones A et B sont toutes deux alphanumériques mais l'une a 50 positions et l'autre 80. Il faut transférer B dans A en utilisant l'ordre : A =B; L'affectation d'une zone alphabétique se fait de la gauche vers la droite, l'ordinateur met dans la zone A le contenu de B en commençant par le début des zones. Dès lors, si la zone A est plus courte que la zone B, l'ordinateur va transférer B dans A de manière à occuper chaque position de A. La partie B qui ne peut passer dans A sera perdue. L'opération A =B; amènera dans A le contenu des positions 1 à 50 de B; le contenu des positions 51 à 80 restera bien entendu dans B mais ne se trouvera jamais dans A. Il y a donc *perte d'information*

dans la zone A.

Si la zone que l'on transpose est plus courte que la zone réceptrice, l'ordinateur garnit de blancs la fin de la zone que l'on affecte.

B = A ;
La totalité de la zone *A* est transmise dans *B*; les positions 51 à 80 de la zone *B* sont mises en blanc.

b) Les zones numériques

DCL X FIXED DEC(5);
DCL Y FIXED DEC(10);

Dans les zones numériques, l'affectation se fait de la *droite vers la gauche* pour la partie entière des nombres et de la *gauche vers la droite* pour la partie décimale. Pour la partie entière, lorsque la zone vers laquelle on transporte l'information est *plus courte*, les chiffres les plus à gauche sont perdus. Pour la partie décimale, ce sont les chiffres les *plus à droite qui sont perdus.*

X = Y;
Le compteur *X* a 5 positions, le compteur *Y* en a 10. Les 5 positions les plus à gauche de *Y* ne sont pas transmises dans *X*. Si *Y* vaut 2434563248, l'opération *X = Y;* donne à *X* la valeur 63248.

Y = X;

Dans le cas où la zone réceptrice est plus longue que la zone émettrice, les positions les plus à gauche de la zone réceptrice sont mises à 0. Si *X* vaut 43256, l'opération *Y = X* donnera à *Y* la valeur 0000043256.

DCL X FIXED DEC(6,2);
DCL Y FIXED DEC(10,4);

X et *Y* sont des zones numériques avec une partie entière et une partie décimale. *X* a 4 positions entières et 2 décimales. *Y* a 6 positions entières et 4 décimales. *Y* vaut, par exemple, 234564,9865. A l'issue de l'opération *X = Y; X* aura pris la valeur 4564,98.

X vaut 5423,45. L'opération *Y = X;* donne à *Y* la valeur 005423,4500. Les zéros sont ajoutés *à gauche* de la partie entière et *à droite* de la partie décimale.

Nous transposerons à présent en PL/1, l'ordinogramme qui correspond à la solution 3 de l'exercice n° 2.

```
1    EX02S03 : PROCEDURE OPTIONS(MAIN);
2    DCL FIC FILE RECORD INPUT;
3    DCL X FIXED DEC(5);
4    DCL 1 ZA,
5         2 ZAA CHAR(1),
6         2 ZAB CHAR(79);
7    DCL 1 ZB,
8         2 ZBA CHAR(1),
9         2 ZBB CHAR(79);
10   ON ENDFILE(FIC) GOTO FIN;
11        READ FILE(FIC) INTO(ZA);
12   AA : X = 0;
13   AB : X = X + 1;
14        READ FILE(FIC) INTO(ZB);
15        IF ZAA = ZBA THEN GOTO AB;
16        PUT EDIT(ZAA,X) (SKIP,A(1), X(2), F(5));
17        ZAA = ZBA;
18        GOTO AA;
19   FIN : PUT EDIT(ZAA,X) (SKIP,A(1), X(2), F(5));
20        END;
```

Cette solution ne demande aucun commentaire. L'instruction d'affectation (17) fait passer la valeur de *ZBA* (1 position) dans *ZAA* (1 position). L'instruction d'écriture permettra de faire apparaître sur une même ligne le code qui représente une modalité de la variable et son nombre d'occurrences. Il y aura entre le code et sa fréquence deux blancs typographiques; ils sont précisés par *X(2)* dans la partie format du *PUT EDIT.*

Exercice 2 solution 4

Cette solution ne demande en principe aucune notion théorique nouvelle. Elle doit être considérée comme une révision de l'ensemble des instructions PL/1 que nous avons rencontrées jusqu'à présent.

```
1    EX02S04 : PROC OPTIONS(MAIN);
2    DCL 1 ZL,
3         2 ZA CHAR(1),
4         2 ZB CHAR(79);
5    DCL FICA FILE RECORD INPUT;
6    DCL FICB FILE RECORD INPUT;
7    ON ENDFILE(FICA) GOTO AA;
8    ON ENDFILE(FICB) GOTO FIN;
9    DCL TA(100) CHAR(1);
10   DCL X(100) FIXED DEC(5) INIT((100)0);
11   L = 0;
12   A1 : READ FILE(FICA) INTO(ZL);
13        L = L + 1;
14        TA(L) = ZA;
15        GOTO A1;
16   AA : READ FILE(FICB) INTO(ZL);
17        J = 0;
18   AB : J = J + 1;
19        IF ZA ¬= TA(J) THEN GOTO AC;
20        X(J) = X(J) + 1;
21        GOTO AA;
22   AC : IF J = L THEN GOTO AA;
23        GOTO AB;
24   FIN : J = 0;
25   BA : J = J + 1;
26        PUT EDIT(TA(J),X(J)) (SKIP,A(1),X(2),F(5));
27        IF J ¬= L THEN GOTO BA;
28        END;
```

On se souvient que l'ordinogramme qui correspond à cette solution est divisé en trois parties. La première consiste à lire les cartes qui portent les codes et à les enregistrer dans la table *TA*. C'est ce que font les instructions 11 à 15. La deuxième partie qui est à proprement parlé le comptage des modalités de la variable comprend les instructions 16 à 21 du programme. Enfin, la troisième partie qui est l'impression des résultats, comporte les opérations 22 à 28.

11. L'instruction DO.

Il est possible de modifier le programme EX02S04 de manière à le rendre un peu moins long et de manière à faire assurer le contrôle de l'indice *J* par une seule instruction qui réaliserait l'initialisation de *J*, sa progression de 1 avant chaque comparaison entre *ZA* et un élément de *TA* et, enfin, l'arrêt du processus d'addition-comparaison lorsque *J* est égal à *L*. Une telle instruction aura le schéma suivant :

```
1   .....
2   DO J = 1 TO L BY 1;
3   .....
4   .....
5   END;
6   .....
```

Les instructions 2 à 5 forment une boucle itérative du programme; elles se répètent autant de fois qu'il faut pour que *J* devienne égal à *L*. Examinons de plus près le travail de l'ordinateur dans un *DO* itératif.

Le contrôle passe de l'instruction 1, quelle qu'elle soit, à l'instruction *DO J = 1 TO L BY 1;* On entre *verticalement* dans la boucle. L'ordinateur commence par initialiser *J* en lui donnant la valeur qui suit le signe égal. Dans ce cas *J* est initialisé à 1. Ensuite, l'ordinateur vérifie si la valeur de *J* n'est pas supérieure à celle de *L*. Si *J* est plus grand que *L*, il quitte la boucle *DO* en passant à l'opération 6. Si *J* est inférieur ou égal à *L*, l'ordinateur exécute les instructions 3, 4 et 5. Le mot *END* que l'on trouve à l'instruction 5 n'est pas la fin du programme mais bien la fin de la boucle *DO;* il équivaut à *un ordre de saut inconditionnel* qui provoque le retour à l'instruction 2. A ce moment, on revient au début de la boucle de manière *horizontale*. L'ordinateur prend alors la valeur qui suit le mot *BY* et il ajoute cette valeur au contenu de l'indice *J*. Après cela, l'ordinateur vérifie à nouveau si *J* n'est pas plus grand que *L* etc...

Le mot *BY* et la valeur qui le suit sont facultatifs lorsque, comme c'est le cas ici, le nombre à ajouter à l'indice est 1. Dès lors, l'expression *DO J = 1 TO L;* équivaut à *DO J=1 TO L BY 1;* l'ordinateur ne trouve pas le mot *BY*, il considère, par défaut, que la valeur à ajouter à chaque itération est 1.

L'instruction *DO* doit s'employer en respectant les quelques règles qui suivent :

- A chaque mot *DO* correspond un mot *END.*
- Il est interdit d'effectuer un saut à partir d'une instruction quelconque du programme se situant hors d'un groupe *DO* vers une opération se situant à l'intérieur d'un groupe *DO*.
- La valeur des indices utilisés dans l'expression *DO J = 1 TO L* ne peut jamais être modifiée après l'entrée dans la boucle *DO*.

Dans une boucle *DO*, on peut rencontrer n'importe quelle instruction y compris un autre *DO*. En voici un exemple :

```
1    DO J = 1 TO 50;
2       DO N = 1 TO 100;
3          .....
4          .....
5       END;
6    END;
```

L'indice utilisé dans la boucle 2 doit être différent de celui employé dans la boucle 1.
Le *END* en 5 correspond au deuxième *DO*, car le *DO* le plus interne est toujours terminé par le *END* le plus interne.
Les instructions 1 et 6 seront exécutées 50 fois à chaque entrée verticale dans la boucle *DO J = 1 TO 50;* les opérations 2 à 5 seront exécutées 5000 fois. En effet, l'entrée verticale dans le deuxième groupe *DO* se produit 50 fois puisque les opérations comprises dans le premier *DO* doivent être exécutées 50 fois. D'autre part, les instructions qui composent le deuxième *DO* doivent être répétées 100 fois à chaque entrée dans le deuxième *DO*. Dès lors, les instructions comprises dans le *DO* le plus interne seront répétées 50 x 100 soit 5000 fois.

Dans le programme EX02S04, on a la possibilité d'employer à deux endroits du programme un *DO* itératif; d'une part, dans la deuxième partie du programme lorsque l'on effectue le dénombrement des codes et, d'autre part, lorsqu'on imprime les résultats du comptage. Voici, à titre d'exemple, le programme EX02S04 modifié en fonction de l'emploi du groupe *DO*. Je ne reprends que les instructions 16 à 28, le début du programme ne subit aucune modification.

```
.....
16   AA : READ FILE(FICB) INTO(ZL);
```

```
17  DO J = 1 TO L;
18      IF ZA ¬= TA(J) THEN GOTO AC;
19     X(J) = X(J) + 1;
20     GOTO AA;
21  AC : END;
22       GOTO AA;
23  FIN : DO J = 1 TO L;
24        PUT EDIT(TA(J),X(J)) (SKIP,A(1),X(2),F(5));
25        END;
26        END;
```

Par rapport à la solution précédente, l'emploi du *DO* permet l'économie de deux instructions. La logique du problème reste bien entendu la même, mais la logique interne du *DO* dispense le programmeur d'initialiser les indices; c'est ce qui explique l'économie d'instructions.

12. Le DO non-itératif.

L'instruction *DO* que nous venons d'étudier sous sa forme la plus simplifiée permet de construire une *boucle itérative* dans un programme. Il existe un autre *DO* qui, lui, sert à construire des boucles de programme non-itératives. Ces boucles peuvent notamment se trouver à la suite des mots *THEN* ou *ELSE*. On sait qu'il n'est pas possible de placer plusieurs instructions à la suite du mot *THEN* ou à la suite du mot *ELSE*. Dans la deuxième et dans la troisième partie d'un IF, on ne peut avoir qu'une seule instruction. Le groupe *DO;.....END;* permet de construire une boucle de programme à la suite de *THEN* ou de *ELSE*. Entre le *DO* et le *END* qui lui est associé, le programmeur a la faculté de placer autant d'instructions qu'il le veut. Voici un exemple d'utilisation du groupe *DO;.....END;* à la suite d'un *THEN*. Je reprends les instructions 16 à 22 de la page précédente.

```
16  AA : READ FILE(FICB) INTO(ZL);
17       DO J = 1 TO L;
18            IF ZA = TA(J) THEN DO;
19                               X(J) = X(J) + 1;
20                               GOTO AA;
21                               END;
```

```
22          END;
23          GOTO AA;
```

L'ordre *DO* n'est suivi d'aucune précision. Il signifie pour l'ordinateur : *EXECUTER LES INSTRUCTIONS QUI SUIVENT JUSQU'A LA RENCONTRE D'UN END.* Il s'agit dans ce cas d'exécuter une seule fois les instructions 19 et 20 après l'entrée du programme dans la boucle formée par *DO;.....END;*

On remarquera que l'emploi de cette boucle apporte trois modifications au programme :

1) Il y a une instruction en plus; en fait, l'ajout des mots *DO* et *END* provoque l'augmentation du nombre d'ordres.

2) La comparaison conditionnelle entre *ZA* et un élément de *TA* se fait de manière positive; l'ordinateur cherche le moment où les deux zones sont égales alors que dans la solution précédente il recherchait l'inégalité. C'est l'emploi du groupe *DO* qui permet d'inverser le sens de la comparaison. En effet, lorsqu'il y a égalité entre *ZA* et un des éléments de *TA*, l'ordinateur doit exécuter deux opérations soit *X(J) = X(J) + 1;* et *GOTO AA;* Ces deux instructions ont pu être placées à l'intérieur d'un groupe *DO;*

3) L'étiquette *AC* a disparu. L'inversion du sens de la comparaison en 18 permet de supprimer l'étiquette *AC*.

La boucle *DO;..... END;* peut se rencontrer simultanément à la suite des éléments *THEN* et *ELSE* dans une même instruction; ce qui donne la structure suivante :

```
IF condition THEN DO;
                  .....
                  .....
                  END;
          ELSE DO;
                  .....
                  .....
                  END;
```

Lorsque la condition est réalisée l'ordinateur exécute les instructions qui se trouvent à l'intérieur du *DO* qui suit *THEN;* si la condition n'est pas réalisée, ce sont les instructions qui suivent le mot *ELSE* qui sont exécutées. Chaque fois le *END* sert de limite à une des boucles.

13. Complément aux déclarations : les tableaux.

Nous avons rencontré dans les exercices précédents des *tables*, c'est-à-dire des zones divisées en un certain nombre d'éléments de type et de longueur identiques.

Un *tableau* est une table à plusieurs dimensions qui se compose d'éléments répartis en lignes et en colonnes. Voici un tableau à deux dimensions :

		X	
	1	2	3
1			
2			
3			
4			
5			
6			

X est un tableau à deux dimensions composé de 18 éléments (6x3), répartis en 6 lignes et 3 colonnes. La déclaration de ce tableau en PL/1 sera :

DCL X(6,3) FIXED DEC(5);

A la suite du nom du tableau (X), on indique entre parenthèses, tout d'abord le nombre de lignes du tableau et ensuite le nombre de colonnes. L'accès à un des éléments se fera grâce à l'emploi de deux indices, le premier désignera le numéro de la ligne où se trouve l'élément que l'on doit atteindre; le second désignera la colonne.

		X	
	1	2	3
1	8	7	5
2	2	0	4
3	32	43	55
4	34	23	67
5	98	39	75
6	43	45	56

X(1,1) = 8	X(1,2) = 7	X(1,3) = 5
X(2,1) = 2	X(2,2) = 0	X(2,3) = 4
X(3,1) = 32	X(3,2) = 43	X(3,3) = 55
X(4,1) = 34	X(4,2) = 23	X(4,3) = 67
X(5,1) = 98	X(5,2) = 39	X(5,3) = 75
X(6,1) = 43	X(6,2) = 45	X(6,3) = 56

L'initialisation d'un tableau se fait ligne par ligne; ainsi, si l'on doit, dans un tableau *X(3,2)*, initialiser les divers éléments comme suit :

	1	2
1	40	20
2	30	50
3	35	45

on écrira :

DCL X(3,2) FIXED DEC(5) INIT(40, 20, 30, 50, 35,45);

Si les éléments (1,1) et (1,2) doivent recevoir 3 comme valeur initiale, les éléments (2,1) et (2,2), 4 et les éléments (3,1) et (3,2), 5,
on écrira :

DCL X(3,2) FIXED DEC(5) INIT(3,3,4,4,5,5);

ou encore en utilisant un facteur de répétition :

DCL X(3,2) FIXED DEC(5) INIT((2)3,(2)4,(2)5);

Les deux déclarations qui précèdent sont équivalentes, elles permettent de définir le tableau suivant :

	X	
	1	2
1	3	3
2	4	4
3	5	5

Si le tableau *X* doit être initialisé comme suit :

	X	
	1	2
1	3	4
2	3	4
3	3	4

On écrira une des déclarations que voici :

DCL X(3,2) FIXED DEC(5) INIT(3,4,3,4,3,4);
DCL X(3,2) FIXED DEC(5) INIT((3) (3,4));

Dans ce dernier exemple, le facteur de répétition (3) porte sur la totalité du groupe (3,4).

Enfin, si l'on veut initialiser tous les éléments de *X* à zéro, on dira :

DCL X(3,2) FIXED DEC(5) INIT((6)0);

L'utilisation des tableaux dans les opérations ne présente aucune difficulté. Ainsi, pour ajouter la valeur 1, par exemple, à l'élément (3,1) du tableau *X*, il suffit d'écrire :

X(3,1) = X(3,1) + 1;

Les chiffres qui donnent le numéro de l'élément peuvent évidemment être remplacés par des indices variables du type *J*. L'opération ci-dessus pourrait être écrite :

X(J,L) = X(J,L) + 1; à condition de donner au préalable la valeur 3 à *J* et la valeur 1 à *L*.

Exercice

Le problème que je propose consiste à répartir une population de plusieurs milliers d'individus en fonction de deux critères; d'une part, la catégorie socio-professionnelle et, d'autre part, le nombre d'enfants.

La catégorie socio-professionnelle est représentée par une des quinze premières lettres de l'alphabet (de A à O); elle est perforée dans la colonne 1 de la carte.

Le nombre d'enfants est une information numérique qui se trouve dans les colonnes 2 et 3 de la carte mécanographique.

Le but du programme est d'établir un tableau de quinze lignes et de six colonnes. La première ligne sera réservée à la catégorie A, la deuxième à la catégorie B, etc. En ce qui concerne les colonnes, on a décidé de répartir le nombre d'enfants en six classes, la première pour les individus sans enfant, la deuxième pour les individus qui ont un enfant, etc. Enfin, la sixième classe regroupera les individus qui ont cinq enfants et plus. Voici le tableau que l'on veut obtenir, nous y avons placé des valeurs imaginaires.

	0	1	2	3	4	5 et +
A	567	345	456	45	34	29
B	323	99	234	234	432	87
C	87	78	98	322	34	23
D	23	45	56	67	54	456
E	26	48	45	56	29	42
F	76	45	32	32	23	23
G	98	67	34	37	42	0
H	56	68	29	56	43	10
I	39	36	28	45	23	15
J	45	56	76	34	9	9
K	65	34	23	26	27	8
L	62	32	34	45	23	15
M	58	28	26	37	15	7
N	24	23	56	23	8	9
O	65	54	34	32	23	19

Solution

a) Les codes des catégories socio-professionnelles seront rangés dans une table alphanumérique composée de quinze éléments d'une position (*TA*). Cette table sera comparée, élément par élément, au contenu de la colonne 1 des cartes. On utilisera

l'indice *J* pour désigner les éléments de la table *TA*.

b) Les données seront lues dans une structure qui s'appellera *ZL* et qui sera divisée en trois parties de niveau 2. Voici la déclaration de *ZL* :

```
DCL 1 ZL,
      2 ZA CHAR(1),
      2 ZB PIC '99',
      2 ZC CHAR(77);
```

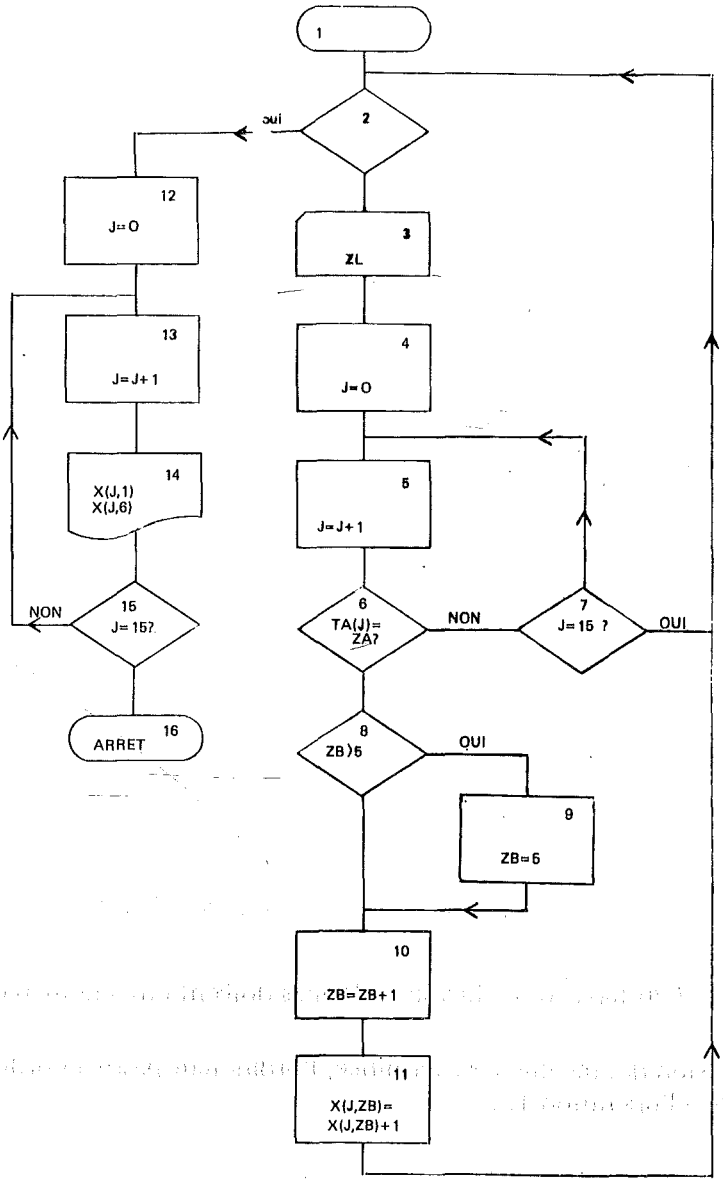
Dans *ZA* on trouvera le code de la catégorie socio-professionnelle. *ZB* contiendra le nombre d'enfants. Cette zone servira d'indice pour le numéro de la colonne. Il faudra toutefois lui faire subir un test pour ramener le nombre d'enfants à cinq lorsqu'un individu a six enfants et plus. En outre, le nombre d'enfants, quel qu'il soit devra être augmenté de 1. En effet, on a vu que les individus sans enfants seront comptabilisés dans la colonne 1, ceux qui ont un enfant dans la colonne 2, etc. Dès lors, pour faire servir *ZB* d'indice, il faut augmenter son contenu d'une unité.

c) Les résultats seront enregistrés dans un tableau numérique de quinze lignes sur six colonnes (*X*).

Le programme contiendra donc les étapes suivantes :

1) Lecture du fichier des données dans la zone *ZL;*
2) Comparaison entre *ZA* et les éléments de *TA* pour repérer la catégorie socio-professionnelle;
3) Contrôle du nombre d'enfants. S'il est supérieur à 5, il sera ramené d'office à 5.
4) Addition de 1 dans un élément du tableau *X*. L'élément sera désigné par l'indice *J* pour le numéro de la ligne et par *ZB* augmenté de 1, pour le numéro de la colonne.
5) Après le traitement de la dernière carte, impression des résultats en tableau.

L'ordinogramme



Commentaires

1) Début du programme.
2) Test de fin de fichier. Après le traitement de la dernière donnée l'ordinateur saute à la figure 12.
3) Lecture d'une carte dans la zone *ZL*.
4) Initialisation de l'indice *J* à 0.
5) Addition de 1 au contenu de *J*.
6) Comparaison entre le contenu de *ZA* et l'élément *J* de *TA*. Puisque chaque élément de *TA* contient un code représentant une catégorie socio-professionnelle, l'identité entre *ZA* et un élément de *TA* donnera le numéro de la ligne du tableau *X* où l'on doit ajouter une occurrence du code. Ainsi, lorsque *ZA* contient la lettre *C*, *ZA* et *TA(J)* seront égaux quand *J* vaudra 3. S'il y a identité entre *TA(J)* et *ZA*, l'ordinateur passe à l'instruction 8, autrement il exécute l'ordre 7.
7) Si *J* vaut 15, l'ordinateur saute à l'ordre 2, autrement il saute à l'opération 5. *J* ne vaudra 15 à l'instruction 7 que dans le cas où le code de *ZA* est autre qu'une des lettres de A à O (cf. à ce sujet l'exercice 2, solution 2, instruction 8).
8) Si *ZB* qui contient le nombre d'enfants est supérieur à 5, l'ordinateur va à l'instruction 9; si ce n'est pas le cas, il passe directement en 10.
9) *ZB* est rendu égal à 5.
10) Addition de 1 au contenu de *ZB*.
11) Addition de 1 à l'élément (*J*,*ZB*) de *X*. Puis retour à l'instruction 2.
12) Initialisation de *J* à 0. C'est ici que commence l'impression des résultats.
13) Addition de 1 à *J*.
14) Impression de la ligne *J* du tableau *X*. Les six colonnes doivent être énumérées une par une.
15) Si *J* vaut 15, l'impression des résultats est terminée, l'ordinateur passe à l'ordre 16, autrement il remonte à l'opération 13.
16) Arrêt.

Le programme PL/1 comprendra deux *DO* itératifs; le premier remplacera les figures 4, 5 et 7 de l'ordinogramme et le second, les figures 12, 13 et 15.

Le programme

```
1  EX : PROC OPTIONS(MAIN);
2  DCL IN FILE RECORD INPUT;
3  ON ENDFILE(IN) GOTO FIN;
4  DCL 1 ZL,
5          2 ZA CHAR(1),
6          2 ZB PIC'99',
7          2 ZC CHAR(77);
8  DCL X(15,6) FIXED DEC(5) INIT((90)0);
9  DCL TA(15) CHAR(1) INIT('A','B','C','D','E','F','G','H','I','J','K',
10                         'L','M','N','O');
11 AA : READ FILE(IN) INTO(ZL);
12       DO J = 1 TO 15;
13            IF TA(J) = ZA THEN DO;
14            IF ZB > 5 THEN ZB = 5;
15            ZB = ZB +1;
16            X(J,ZB) = X(J,ZB) +1;
17            GOTO AA;
18            END;
19       END;
20       GOTO AA;
21 FIN : DO J = 1 TO 15;
22            PUT EDIT (X(J,1),X(J,2),X(J,3),X(J,4),X(J,5),X(J,6))
23                     (SKIP,F(5),X(2),F(5),X(2),F(5),X(2),F(5),
24                     X(2),F(5),X(2),F(5));
25       END;
26       END;
```

Remarques :

1) La table alphanumérique *TA* reçoit comme valeur initiale les quinze premières lettres de l'alphabet dans l'ordre alphabétique normal à raison d'un caractère par

élément. De la sorte *TA(1)* contiendra *A*, *TA(2)* contiendra *B* et *TA(15)* contiendra *O*.

2) La partie principale du programme durant laquelle l'ordinateur effectue le dénombrement des modalités de la variable *ZA*, est entièrement contenue dans un *DO* itératif (instruction 12); le *END* qui correspond à ce *DO* se trouve à la ligne 19. L'indice *J* est initialisé à 1 lors de l'entrée dans la boucle *DO* et augmenté de 1 à chaque saut de l'opération 19 vers l'opération 12.

   L'indice *J* sert à comparer successivement chacun des éléments de la table *TA* à la zone *ZA*. En outre, lorsqu'il y a identité entre *ZA* et un élément de *TA*. l'indice *J* est utilisé pour désigner la ligne du tableau *X* à laquelle on doit ajouter la valeur 1. Au moment où *ZA* et un élément de *TA* sont égaux, l'ordinateur rencontre un deuxième *DO* (ligne 13). Il s'agit dans ce cas d'un *DO* non itératif qui se termine au *END* de la ligne 18. Ce *DO* et le *END* correspondant englobent les instructions que l'ordinateur doit exécuter lorsque *ZA* et *TA(J)* sont identiques.

3) Les instructions 15 et 16 pourraient être remplacées par la seule opération suivante :

   X(J,ZB + 1) = X(J,ZB + 1) + 1;

   Cet ordre signifie que l'ordinateur doit ajouter la valeur 1 à l'élément du tableau *X* dont les coordonnées sont, d'une part, la ligne qui correspond à la valeur *J* et, d'autre part, la colonne qui est désignée par la *valeur de ZB augmentée de 1.* En réalité, l'ordinateur va, pour exécuter cet ordre, commencer par résoudre l'expression *ZB + 1* puis il ajoutera la valeur 1 à l'élément de *X*.

4) L'instruction 20, *GOTO AA*; ne s'exécute que si *ZA* contient un caractère autre qu'une des quinze premières lettres de l'alphabet.

5) L'ordre d'écriture permet d'imprimer en une fois une ligne entière du tableau *X* puisque les six éléments de *X* sont énumérés. Cet ordre est long à écrire; il peut être abrégé grâce à l'utilisation de deux indices et de deux *DO* itératifs imbriqués. Les instructions seront dans ce cas :

```
21  FIN : DO J = 1 TO 15;
22              PUT SKIP;
23              DO L = 1 TO 6;
```

```
24                  PUT EDIT(X(J,L)) (X(2),F(5));
25              END;
26          END;
```

Ce groupe d'instructions est exécuté de la manière suivante : l'ordinateur commence par initialiser l'indice *J* à 1, ensuite, il exécute un saut du papier à l'imprimante qui est ainsi positionnée à une nouvelle ligne. Après le saut du papier, l'ordinateur initialise l'indice *L* à 1 puis il imprime l'élément de *X* dont les coordonnées correspondent à *J* pour la ligne et à *L* pour la colonne.

L'expression *X(2)*qui se trouve dans la deuxième partie du *PUT EDIT* fait que l'ordinateur laisse deux positions blanches sur le papier avant d'imprimer la valeur contenue dans un élément du tableau. Les opérations 23, 24 et 25 sont exécutées 6 fois pour chacune des valeurs que *J* peut prendre de sorte que l'instruction d'écriture sera répétée six fois pour chaque valeur de *J* tandis que l'indice *L* sera modifié.

Le *END* de la ligne 25 correspond au *DO* de la ligne 23 et le *END* de la ligne 26 au *DO* de la ligne 21.

Quelle que soit la solution utilisée pour imprimer les résultats, le tableau que l'on obtient est un tableau aveugle sans titre ni pour les lignes, ni pour les colonnes. Si l'on veut un tableau avec des titres, il faut commencer par imprimer au-dessus de chaque colonne un chiffre qui correspond au nombre d'enfants comptabilisés dans chacune des colonnes. D'autre part, pour les titres des lignes, on imprimera au moment du *PUT SKIP* que l'on trouve dans la solution 2, l'élément de *TA* qui est désigné par la valeur de *J*.

```
1  FIN : PUT EDIT('0','1','2','3','4','5' et  +') (SKIP,X(7),A(1),X(6),
2                  A(1),X(6),A(1),X(6),A(1),X(6),A(1),X(6),A(6));
3       DO J = 1 TO 15;
4            PUT EDIT(TA(J)) (SKIP,A(1));
5            DO L = 1 TO 6;
6                 PUT EDIT (X(J,L)) (X(2),F(5));
7            END;
8       END;
```

Le premier *PUT EDIT* imprime le titre des colonnes, c'est-à-dire le nombre d'enfants

par individu. C'est une instruction non-répétitive. L'opération de la ligne 4 a un double rôle; elle provoque, grâce à l'option *SKIP*, l'avancement du papier sur l'imprimante et, d'autre part, elle fait apparaître en début de ligne le code qui se trouve dans l'élément de *TA* spécifié par *J* et qui correspond à la ligne du tableau *X* que l'ordinateur va imprimer. Cette instruction est répétée 15 fois.

Le troisième *PUT EDIT* est destiné à imprimer l'élément du tableau *X* dont les coordonnées sont *J* et *L*. Ce *PUT EDIT* est exécuté 90 fois.

14. Les fonctions incorporées du PL/1.

Après ces quelques exercices sur le traitement des informations enregistrées en longueur fixe, nous passerons à l'étude des instructions qui permettent de traiter des données de longueur variable.

Dans la deuxième partie du chapitre consacré à l'ordinogramme, j'ai donné des exemples du traitement des informations linguistiques. Nous reprendrons les trois problèmes pour les transposer en PL/1.

Dans le premier problème, nous avons vu que pour rechercher un mot dans une zone alphanumérique, l'ordinateur doit parcourir la zone, position par position du début jusqu'à la fin. Reprenons l'exemple que nous avions donné précédemment.

Qui aime bien châtie bien

Ce texte a été perforé sur une carte mécanographique à partir de la colonne 1. Le programme devra lire les données et les stocker dans une zone de lecture déclarée comme suit :

DCL ZL CHAR(80);

L'opération de lecture transporte dans *ZL* le contenu de la carte; le texte commence donc à la première position de *ZL*.

```
                  1         2         2         8
    1             0         0         6         0
ZL  QUI AIME BIEN CHATIE BIEN
```

Pour progresser de la position 1 à la position 80 de *ZL*, on peut utiliser un *DO* itératif (*DO J = 1 TO 80;*) dont l'indice servira à désigner la position de *ZL* sur laquelle on doit travailler. On parlera donc dans le programme de la position *J* de *ZL*. Cependant le langage PL/1 ne possède pas d'instruction pour atteindre, dans une zone alphanumérique, une position quelconque de cette zone. En effet, on ne peut pas, par exemple, écrire *ZL(J)* car ce type de notation est exclusivement réservé aux tables, ce que n'est pas *ZL*. Pour rendre possible le traitement des zones alphabétiques, les créateurs du langage PL/1 ont ajouté aux instructions de base que nous avons étudiées jusqu'à présent des "fonctions", c'est-à-dire des petits programmes capables de réaliser des opérations complexes; ce sont les *fonctions incorporées* du langage PL/1. De manière générale, l'utilisation de ces fonctions est simple; elle se fait grâce au nom de la fonction et grâce à quelques indications que l'on appelle des arguments.

Il existe en PL/1 deux types de fonctions, les fonctions mathématiques et les fonctions propres au traitement des données de longueur variable. Ce sont ces dernières que nous examinerons.

A. La fonction INDEX (deux arguments).

Cette fonction permet de *localiser* dans une zone la position où se trouve un caractère ou une suite de caractères. Les deux arguments sont, d'une part, le nom de la zone dans laquelle on recherche une information et, d'autre part, l'information que l'on recherche. La fonction *INDEX* donne le résultat suivant : elle détermine la position de départ d'une suite de caractères et attribue à un indice du type *J* une valeur qui donne le numéro de la position de la séquence recherchée dans la zone. Ainsi, pour trouver le mot *aime* dans la zone *ZL* qui contient *"qui aime bien châtie bien"*, on doit écrire l'instruction suivante :

```
J = INDEX(ZL,'AIME');
```

ce qui signifie pour l'ordinateur :
DONNER A L'INDICE J UNE VALEUR EGALE A LA POSITION DE LA ZONE ZL OU COMMENCE LA SEQUENCE DE CARACTERES "AIME".

Après cette opération la valeur de *J* sera 5 puisque le mot *"aime"* commence à la cinquième position de *ZL*.

Lorsque l'argument recherché ne se trouve pas dans la zone, la fonction *INDEX attribue la valeur 0 à l'indice.* Ainsi, si l'on exécute l'opération *J = INDEX(ZL,'AIMER');* l'indice *J* prendra la valeur 0 puisque la séquence "AIMER" est absente de la zone *ZL*.

Il est important de noter que la fonction *INDEX* repère uniquement la première séquence de caractères qu'elle rencontre dans une zone de sorte que si l'on voulait, par exemple, compter le nombre de fois qu'une suite de caractères apparaît dans une zone, on ne pourrait pas employer la fonction *INDEX*. Si on voulait compter le nombre de fois que le mot BIEN est employé dans la zone *ZL*, il ne serait pas possible de le faire avec la fonction *INDEX* : l'instruction *J = INDEX(ZL,'BIEN');* donnera toujours à *J* la valeur 10. Ce résultat est parfaitement logique puisque la fonction *INDEX* commence normalement à parcourir une zone à la première position et qu'elle s'arrête dès qu'elle a repéré l'information recherchée. Pour atteindre la seconde occurrence du mot BIEN dans la zone *ZL*, il faut une fonction qui permet de commencer l'examen d'une zone à une position déterminée par un indice, cette fonction s'appelle, en PL/1, *SUBSTR.*

B. La fonction SUBSTR (deux ou trois arguments).

Cette fonction permet *d'extraire* le contenu d'une zone à partir d'une position précisée par une valeur fixe ou variable. La longueur de l'information extraite est définie par un nombre fixe ou variable.

Lorsque la fonction *SUBSTR* est utilisée avec trois arguments, on doit trouver dans l'ordre :

1) Le nom de la zone d'où l'on doit extraire des informations;
2) Un nombre qui indique le numéro de la position à partir de laquelle doit commencer l'extraction;
3) Un nombre qui indique combien de positions doivent être extraites.

On veut transmettre le mot AIME qui commence à la position 5 de *ZL* et qui compte 4 lettres, dans une zone *ZA* déclarée *CHAR(4).* Les instructions seront les suivantes :

```
DCL ZL CHAR(80);
DCL ZA CHAR(4);
ZA = SUBSTR(ZL,5,4);
```

Cette dernière instruction signifie : *TRANSPORTER, DANS LA ZONE ZA, 4 POSITIONS DE LA ZONE ZL A PARTIR DE LA POSITION 5.*

Les deuxième et troisième arguments peuvent être des variables, c'est-à-dire des indices numériques du type *J*.

```
DCL ZL CHAR(80);
DCL ZA CHAR(4);
J = 5;
ZA = SUBSTR(ZL,J,4);
```

Cette dernière instruction transmet le mot AIME dans *ZA*; le deuxième argument *(J)* est un indice variable auquel on peut donner toutes les valeurs comprises entre 1 et le nombre maximum de positions de la zone *ZL*.

```
DCL ZL CHAR(80);
DCL ZA CHAR(4);
J = 5;
L = 4;
ZA = SUBSTR(ZL,J,L);
```

Cet ordre donne le même résultat que les deux *SUBSTR* précédents : le mot AIME est transmis dans la zone *ZA*. Les arguments 2 et 3 sont des indices numériques variables.

Lorsque la fonction *SUBSTR* est employée avec *deux arguments* seulement, on trouve :
1) Le nom de la zone d'où on extrait des informations;
2) Un nombre fixe ou variable qui indique le numéro de la position à partir d'où commence l'extraction.

Le troisième argument est défini automatiquement par l'ordinateur qui extrait de la zone tout ce qui se trouve entre la position déterminée par le deuxième argument et la fin de la zone. *ZL* contient la phrase *qui aime bien châtie bien;* on doit transmettre dans une zone *ZA* déclarée *CHAR(80)* le contenu de *ZL* à partir de la position 5.

```
DCL ZL CHAR(80);
DCL ZA CHAR(80);
ZA = SUBSTR(ZL,5);
```

Après cette opération, la zone *ZA* contient *"aime bien châtie bien"*; l'absence du troisième argument de la fonction *SUBSTR* fait que tout le contenu de *ZL*, à partir de la position 5, est transporté dans *ZA*; la fin de la zone *ZA* est mise à blanc.

Dans le cas où la zone réceptrice est plus courte que la zone émettrice, la zone réceptrice est remplie à partir de la gauche et les informations qui se trouvent à droite dans la zone émettrice sont perdues.

```
DCL ZA CHAR(4);
DCL ZL CHAR(80);
ZA = SUBSTR(ZL,5);
```

L'ordinateur doit transporter le contenu de *ZL* à partir de la position 5 dans la zone *ZA*. Comme cette dernière n'a que 4 positions, elle contiendra uniquement le mot AIME.

Nous allons à présent utiliser les fonctions *INDEX* et *SUBSTR* pour réaliser le programme relatif au traitement des données linguistiques qui consiste à rechercher les apparitions d'un mot dans un contexte déterminé. Je propose trois solutions, la première est fondée sur l'emploi de la fonction *INDEX*, la deuxième sur l'utilisation de *SUBSTR* et la troisième sur l'emploi combiné des deux fonctions.

Solution 1

```
1  EX : PROC OPTIONS(MAIN );
2  DCL ZL CHAR(80);
3  DCL IN FILE RECORD INPUT;
4  ON ENDFILE(IN) GOTO FIN;
5  AA : READ FILE(IN) INTO(ZL);
6        J = INDEX(ZL,'BIEN');
7        IF J ¬= 0 THEN PUT EDIT(ZL) (SKIP,A(80));
8        GOTO AA;
9        FIN : END;
```

Après l'opération de lecture en 5, l'ordinateur cherche dans la zone *ZL* la position où commence le mot BIEN. Si ce mot ne se trouve pas dans *ZL, l'indice J prend la valeur 0.* L'ordinateur vérifie ensuite si *J* est différent de 0 ce qui équivaut à voir si le mot BIEN est employé dans *ZL.* Si *J* vaut 0, l'ordinateur passe directement à l'ordre 8, autrement il imprime le contenu de la zone de lecture.

Cette solution a deux qualités évidentes, sa simplicité et sa brièveté. Toutefois, elle ne peut convenir pour trouver toutes les occurrences d'un mot dans un texte puisque, je le rappelle, la fonction *INDEX* s'arrête toujours à la première apparition dans une donnée de l'élément recherché. La fonction *SUBSTR* va permettre de repérer tous les

emplois d'un mot dans un contexte.

Solution 2

```
1  EX : PROC OPTIONS(MAIN);
2  DCL ZL CHAR(80);
3  DCL IN FILE RECORD INPUT;
4  ON ENDFILE(IN) GOTO FIN;
5  AA : READ FILE(IN) INTO(ZL);
6        J = 0;
7  AB : L = J + 1;
8  AC : J = J + 1;
9        IF SUBSTR(ZL,J,1) >= 'A' THEN GOTO AC;
10       IF SUBSTR(ZL,L,J-L) = 'BIEN' THEN PUT EDIT(ZL)
           (SKIP,A(80));
11       IF J = 76 THEN GOTO AA;
12       GOTO AB;
13 FIN : END;
```

Cette solution suit l'ordinogramme que nous avons réalisé dans le chapitre précédent mais les instructions 9 et 10 demandent peut-être quelques mots d'explications.

L'ordre 9 consiste à vérifier si la position de *ZL* désignée par l'indice *J* contient un caractère alphabétique. Si c'est le cas, l'ordinateur remonte à l'instruction 8.

Lorsqu'une position de *ZL* contient un caractère plus petit que la lettre A cela signifie que l'indice *J* désigne une position de *ZL* qui suit la fin d'un mot. L'ordinateur exécute alors l'opération 10. La première partie de cette ligne signifie :

SI LE CONTENU DE LA ZONE ZL A PARTIR DE LA POSITION L SUR UNE LONGUEUR DE J - L POSITIONS EST LE MOT"BIEN" ALORS ...

L'expression *J - L* se comprend mieux si l'on reprend la représentation graphique de la zone *ZL*.

```
                          1         2         2
          4         9     4         1         6
ZL  |QUI AIME BIEN CHATIE BIEN                  |
```

Voici les différentes valeurs des indices au moment où l'ordinateur identifie une fin de mot. Je rappelle que lorsque *J* a une valeur qui coïncide avec la fin d'un mot, *L*, quant à lui, a une valeur qui désigne la première position de ce mot.

mot	valeur de *L*	valeur de *J*	valeur de *J* - *L*
QUI	1	4	3
AIME	5	9	4
BIEN	10	14	4
CHATIE	15	21	6
BIEN	22	26	4

On voit dès lors que l'expression *J* - *L* donne comme résultat une valeur égale au nombre de lettres de chaque mot du texte.

Solution 3

Il existe une troisième solution possible pour ce problème; elle consiste à combiner l'emploi des fonctions *INDEX* et *SUBSTR* dans une même opération.

```
1  EX : PROC OPTIONS(MAIN);
2  DCL ZL CHAR(80);
3  DCL IN FILE RECORD INPUT;
4  ON ENDFILE(IN) GOTO FIN;
5  AA : READ FILE(IN) INTO(ZL);
6       J = 1;
7       L = 0;
8  AB : L = L + J;
9            J = INDEX(SUBSTR(ZL,L),'BIEN');
10           IF J = 0 THEN GOTO AA;
11           PUT EDIT(ZL) (SKIP,(80));
12          GOTO AB;
13 FIN : END;
```

L'expression de la ligne 9 signifie que l'ordinateur doit localiser le mot BIEN dans la *partie de la zone ZL qui commence à la position désignée par L* et qui va jusqu'à la fin de la zone. Voici ce que fait l'ordinateur pour la phrase *qui aime bien châtie bien.*

Instruction		valeur des indices et de *ZL*
5	Lecture dans *ZL*	ZL = 'QUI AIME BIEN CHATIE BIEN'
6		J = 1
7		L = 0
8	*L = L + J;*	L = 1
9a	Le *SUBSTR(ZL,L)* = 'QUI AIME BIEN CHATIE BIEN' puisque *L = 1.*	
9b	*J = 10* : la première occurrence du mot BIEN commence à la position 10 de *ZL*.	
10	Puisque *J* est différent de 0, l'ordinateur passe à l'ordre 11.	
11	Impression de la zone *ZL*.	
12	Retour à *AB* (instruction 8).	
8	*L = L + J;*	L = 1 + 10 soit 11.
9a	L'expression *SUBSTR(ZL,L)* est égale au contenu de la zone *ZL* à partir de la onzième position jusqu'à la fin de la zone; *ZL = 'IEN CHATIE BIEN '.*	
9b	La valeur de J sera 12 car l'ordinateur commence à parcourir la zone *ZL* à la position désignée par *L en considérant que cette position est la première de la zone ZL.*	
10	*J* n'est pas égal à 0, l'ordinateur exécute l'opération 11.	
11	Impression du contenu de *ZL.*	
12	Retour à *AB.*	
8	*L = L + J;*	L = 11 + 12 soit 23.
9a	Le *SUBSTR(ZL,L)* désigne cette fois la partie de la zone *ZL* qui va de la vingt-troisième position à la fin. *ZL = 'IEN '.*	
9b	La fin de la zone *ZL* ne contient plus le mot BIEN, c'est pourquoi l'ordinateur donne à *J* la valeur 0.	
10	Retour à *AA* (instruction 5) pour lire une nouvelle donnée.	

Pour bien comprendre la manière dont l'ordinateur exécute l'expression *J = INDEX(SUBSTR(ZL,L),'BIEN');* il n'est pas inutile de décomposer cette instruction.

Le premier argument de la fonction *INDEX* est constitué par l'expression *SUBSTR(ZL, L);* l'ordinateur commence par évaluer cette expression : pour cela il extrait de *ZL* tout ce qui se trouve à partir de la position désignée par l'indice *L*. Après cela, l'ordinateur prend le second argument de la fonction *INDEX* (le mot BIEN) et cherche la position à laquelle il est situé dans la portion de *ZL* qu'il doit parcourir. Il est important de noter qu'avant d'exécuter la fonction *INDEX*, l'ordinateur attribue à l'indice *J*

la valeur 0 et qu'il considère que la position de *ZL* définie par l'indice *L* est la première position de la zone. On comprend dès lors qu'il donne à *J*, lors du deuxième passage dans la séquence d'instructions 8, 9, 10, la valeur 12. Ce nombre est en réalité *la distance qui sépare la deuxième lettre du premier mot* BIEN *de la première lettre de la deuxième occurrence de ce mot.*

La combinaison des fonctions *INDEX* et *SUBSTR* semble peut-être compliquer inutilement la logique du programme. En effet, on aurait pu décomposer l'instruction 9 en deux opérations qui donnent des résultats identiques à ceux que nous avons obtenus et qui n'enlèvent rien aux performances du programme. Le seul inconvénient de ce procédé est qu'il nécessite la déclaration d'une zone de travail supplémentaire (la zone ZB dans l'exemple qui suit).

```
1  EX : PROC OPTIONS(MAIN);
2  DCL ZL CHAR(80);
3  DCL ZB CHAR(80);
4  DCL IN FILE RECORD INPUT;
5  ON ENDFILE(IN) GOTO FIN;
6  AA : READ FILE(IN) INTO(ZL);
7        J = 1;
8        L = 0;
9  AB : L = L + J;
10       ZB = SUBSTR(ZL,L);
11       J = INDEX(ZB,'BIEN');
12       IF J = 0 THEN GOTO AA;
13       PUT EDIT(ZL) (SKIP,A(80));
14       GOTO AB;
15       END;
```

La zone *ZB* est destinée à recevoir la partie de *ZL* qui commence à la position *L*. Le contenu de *ZB* sera donc successivement : 1) QUI AIME BIEN CHATIE BIEN
2) IEN CHATIE BIEN
3) IEN

On le voit, cette solution est pratiquement identique à la précédente puisque le calcul de la valeur des indices *L* et *J* reste le même. On comprend peut-être mieux à présent pourquoi l'indice *J* vaut successivement 10, 12 et 0.

L'emploi des fonctions *INDEX* et *SUBSTR* dans des instructions distinctes permet d'écrire un programme plus simple que les précédents où l'on a besoin d'un seul indice, d'une zone de lecture (*ZL*) et d'une zone de travail (*ZB*).

```
1  EX : PROC OPTIONS(MAIN);
2  DCL IN FILE RECORD INPUT;
3  ON ENDFILE(IN) GOTO FIN;
4  DCL ZL CHAR(80);
5  DCL ZB CHAR(80);
6  AA : READ FILE(IN) INTO(ZL);
7       ZB = ZL;
8  AB : J = INDEX(ZB,'BIEN');
9       IF J = 0 THEN GOTO AA;
10      PUT EDIT(ZL) (SKIP,A(80));
11      ZB = SUBSTR (ZB, J + 1);
12      GOTO AB;
13 FIN : END;
```

Le procédé employé ici consiste à faire glisser le contenu de la zone *ZB* de la droite vers la gauche (opération 11) en éliminant à chaque déplacement ce qui se trouve avant la position *J + 1*. Autrement dit, l'ordinateur transmet au début de la zone *ZB* le contenu de cette zone à partir de la position *J + 1*.

Ces quelques solutions du même problème nous ont permis l'utilisation des fonctions *INDEX* et *SUBSTR;* nous allons les rencontrer dans les deux solutions de l'exercice qui consiste à préparer un index inverse d'un fichier de mots.

Je rappelle que la première manière de constituer un index inverse se déroule en deux phases : la zone où le mot est enregistré est parcourue une première fois de gauche à droite jusqu'à la fin du mot puis une seconde fois de droite à gauche pour transmettre le mot, caractère par caractère, en l'inversant.

Les données sont lues dans une structure *ZL* dont les 20 premières positions contiennent les mots; la zone d'inversion s'appelle *ZB*.

```
1  S01 : PROC OPTIONS (MAIN);
2  DCL 1 ZL,
```

```
 3          2 ZA CHAR(20).
 4          2 YA CHAR(60);
 5  DCL IN FILE RECORD INPUT;
 6  ON ENDFILE(IN) GOTO FIN;
 7  DCL ZB CHAR(20);
 8  AA : READ FILE(IN) INTO(ZL);
 9       ZB = ' ';
10       DO J = 1 TO 20;
11       IF SUBSTR(ZA,J,1) = ' ' THEN GOTO AB;
12       END;
13  AB : DO L = 1 TO 20;
14       J = J - 1;
15       SUBSTR(ZB,L,1) = SUBSTR(ZA,J,1);
16       IF J = 1 THEN GOTO AC;
17       END;
18  AC : PUT EDIT(ZB) (SKIP,A(20));
19       GOTO AA;
20  FIN : END;
```

L'instruction 9 assure la mise à blanc de la zone *ZB*. Elle est destinée à réinitialiser la zone avant le traitement d'un mot. La raison pour laquelle on doit remettre *ZB* à blanc est la suivante : la transmission des mots dans la zone *ZB* se fait lettre par lettre de sorte que l'ordinateur modifie uniquement les positions de *ZB* qui doivent recevoir une lettre. Lorsque l'ordinateur doit traiter les mots *qui aime bien châtie bien*, perforés chacun sur une carte, voici ce qui se passerait si la zone *ZB* n'était pas remise à blanc.

Contenu de *ZA*	Contenu de *ZB*
position n°	positon n°
1 2 3 4 5 6 7 8 9 10	1 2 3 4 5 6 7 8
QUI	IUQ
AIME	EMIA
BIEN	NEIB
CHATIE	EITAHC
BIEN	NEIBHC

L'absence de réinitialisation provoque des erreurs lorsqu'un mot est plus court que celui qui le précède : c'est le cas de la deuxième occurrence du mot BIEN; les deux premières lettres de CHATIE subsistent à la suite du mot BIEN.

L'opération 11 permet de détecter la position qui suit la dernière lettre d'un mot. Pour QUI, l'indice *J* vaut 4 au moment où la position *J* de *ZA* contiendra un blanc; l'ordinateur passera alors à la deuxième partie du programme (instructions 13 à 17) qui consiste à ranger le mot dans la zone *ZB* en l'inversant.

En 13, l'indice *L* vaut 1 au départ de la boucle *DO*, il va croître de 1 à chaque répétition des instructions 13 à 17.

L'opération 15 fait intervenir deux fois la fonction *SUBSTR*. A la droite du signe égal, l'emploi de *SUBSTR* est normal, nous en avons parlé suffisamment. Par contre, l'utilisation de *SUBSTR* avant le signe égal appelle quelques observations.

Tout d'abord, dans cette position,*SUBSTR* joue le rôle d'une *pseudo-variable*, cela signifie que l'expression *SUBSTR(ZB,L,1)* se comporte comme une zone dans laquelle on transmet des informations. En effet, il n'est pas question d'extraire de *ZB* des données mais bien de transporter des lettres de *ZA* vers *ZB*. Les arguments de *SUBSTR* employé comme pseudo-variable sont les mêmes que ceux de *SUBSTR* utilisé comme fonction d'extraction. Leur signification est la suivante :

1) nom de la zone dans laquelle on doit transmettre une information;
2) numéro de la position à partir de laquelle les informations doivent être rangées;
3) nombre de positions que l'on occupera dans la zone; cet argument est facultatif.

La seconde observation que l'on doit faire au sujet de l'emploi d'une fonction comme pseudo-variable est que la fonction *INDEX* ne peut être placée à la gauche d'un signe égal. *INDEX* ne peut être une pseudo-variable.

La deuxième manière de constituer un index inverse consiste à parcourir la zone de lecture de la droite vers la gauche et à ne transmettre dans la zone *ZB* que les caractères qui sont différents du blanc.

```
1  S02 : PROC OPTIONS(MAIN);
2  DCL 1 ZL,
3         2 ZA CHAR(20);
4         2 YA CHAR(60);
```

```
 5  DCL IN FILE RECORD INPUT;
 6  ON ENDFILE(IN) GOTO FIN;
 7  DCL ZB CHAR(20);
 8  AA : READ FILE(IN) INTO(ZL);
 9       L = 1;
10       ZB = ' ';
11       DO  J = 20 TO 1 BY -1;
12            IF SUBSTR(ZA,J,1) ¬= ' ' THEN DO;
13                 SUBSTR(ZB,L,1) = SUBSTR(ZA,J,1);
14                 L = L + 1;
15            END;
16       END;
17       PUT EDIT(ZB) (SKIP,A(20));
18       GOTO AA;
19  FIN : END;
```

Cette solution ne demande guère de commentaires. L'ordinateur parcourt la zone *ZA* en commencant par la fin *(DO J = 20 TO 1 BY -1;).* Si la position *J* de *ZA* ne contient aucune information, l'ordinateur va diminuer immédiatement *J* de 1. Si la position *J* n'est pas blanche, elle est transmise dans la zone *ZB* et *L* qui désigne au départ du programme la première position de *ZB(L = 1)* est augmenté de 1.

15. Les zones de longueur variable.

A. Déclaration.

Après avoir fait ces quelques exercices que nous avions rencontrés dans le chapitre des ordinogrammes, nous allons étudier les zones de longueur variable. Nous venons de voir comment traiter une information de longueur variable rangée dans une zone fixe. Le PL/1 possède un type de déclaration et des instructions qui permettent de traiter la donnée de longueur variable dans une zone qui s'adapte automatiquement à cette longueur. Ce type de zone est si j'ose dire "élastique". La dimension minimum d'une zone variable est 0 octet et sa dimension maximum est fixée au moment de la déclaration. Enfin, on doit trouver dans la définition d'une zone de longueur variable le mot *VARYING* ou son abréviation *VAR.*

DCL ZL CHAR(500) VAR;

La zone *ZL* pourra avoir 500 positions au maximum. *VAR* indique que la longueur de *ZL* s'adaptera aux informations qui y seront rangées.

ZL = 'QUI AIME BIEN CHATIE BIEN';

Cette instruction range dans *ZL* la phrase *qui aime bien châtie bien*; simultanément le contenu précédent de *ZL* (s'il y en avait un) est effacé et *ZL* devient une zone de 25 positions. Le programmeur a utilisé la zone variable *ZL* comme une autre zone; c'est l'ordinateur qui a contrôlé l'ensemble de l'opération et qui connaît la longueur réelle de la zone. Si le programmeur désire savoir le nombre d'octets de *ZL*, il devra utiliser une fonction appelée *LENGTH*, c'est-à-dire *LONGUEUR* qui note dans un indice le nombre de positions d'une zone variable.

B. La fonction LENGTH

J = LENGTH(ZL);

Pour le contenu que nous avons attribué à *ZL* précédemment, la valeur de *J* sera 25. *LENGTH* est une fonction à un seul argument : c'est le nom de la zone dont on veut évaluer la longueur.

C. Manipulation des zones variables

Comment transmettre des informations dans une zone variable ?

1.- Lorsque les données à placer dans une zone de longueur variable sont des constantes (*'qui aime bien châtie bien'*), il suffit d'écrire une instruction semblable à celle que nous avons rencontrée plus haut.

   ZL = 'QUI AIME BIEN CHATIE BIEN';

2.- Si on doit transmettre des blancs dans une zone *VARYING*, il faut savoir que :
   a) *ZL* = 'b'; transmet *un seul* blanc dans la zone *ZL* qui de ce fait est ramenée à une position quel que soit son contenu antérieur.
   b) La transmission de plusieurs blancs ne peut donc se faire qu'en indiquant à l'ordinateur le nombre de blancs que l'on veut : *ZL = (500)'b';*

3.- Lorsqu'on doit transférer le contenu d'une zone de longueur fixe dans une zone variable, il faut indiquer à l'ordinateur le nombre de positions qu'il doit prendre,

autrement il transporte dans la zone variable la totalité de la zone fixe, positions blanches comprises. En voici un exemple :

```
DCL YL CHAR(80);
DCL ZL CHAR(500) VAR;
YL = 'QUI AIME BIEN CHATIE BIEN';
```

YL qui est une zone de longueur fixe (80 positions), contient dans les positions 1 à 25 le texte et dans les positions 26 à 80 des blancs.

YL QUI AIME BIEN CHATIE BIENbbbbbbbbbbbbbbbbbbbbbbb

Le caractère b représente des blancs.

ZL = YL;
Le contenu de *YL* est transmis dans la zone variable *ZL* qui devient :

ZL QUI AIME BIEN CHATIE BIENbbbbbbbbbbbbbbbbbbbbbbb

Par conséquent, la longueur de *ZL* est de 80 positions et le texte de 25 caractères est suivi de 55 positions blanches. L'instruction *ZL = YL* range donc une zone de longueur fixe dans une zone de longueur variable. Or, ce que l'on doit faire c'est transmettre dans *ZL* uniquement le texte et non les blancs qui le suivent. Il faut pour cela trouver une instruction qui permette de savoir à quelle position de *YL* le texte se termine. On sait que la fonction *INDEX* est destinée à localiser dans une zone alphanumérique la position à laquelle commence une séquence déterminée de caractères. L'absence de caractères peut être représentée par une suite de caractères blancs; dès lors la fonction *INDEX* aura comme deuxième argument deux blancs.

J = INDEX(YL,'bb');
La valeur de *J* sera 26 puisque c'est à la position 26 que commence la première séquence de deux caractères blancs. On peut alors écrire :

ZL = SUBSTR(YL,1,J - 1);

Ce qui signifie que l'ordinateur va transporter dans *ZL* les informations qui vont de la position 1 à la position 25 de *YL*. Les deux instructions qui précèdent peuvent être remplacées par :

ZL = SUBSTR(YL,1,INDEX(YL,'bb')-1);

L'ordinateur commence par évaluer l'expression *INDEX(YL,'bb')-1* pour laquelle il trouve la valeur 25; ensuite, il transporte dans *YL* le contenu de *ZL* à partir de la position 1 jusqu'à la position 25.

Dans cette instruction, on trouve deux ouvertures et deux fermetures de parenthèses. La première parenthèse ouverte et la deuxième parenthèse fermée englobent les arguments de la fonction *SUBSTR;* la deuxième parenthèse ouverte et la première fermée contiennent les arguments d'*INDEX.*

Au terme de l'opération qui précède, la zone *ZL*, longue de 25 positions, contient la phrase complète qui se trouvait dans *YL*.

16. La concaténation.

Il y a en PL/1 une opération qui est particulièrement bien adaptée au traitement des zones de longueur fixe : c'est l'instruction *de concaténation.*

La concaténation permet de juxtaposer dans une zone réceptrice le contenu de plusieurs zones émettrices dans l'ordre où ces zones sont énumérées. Cette opération a pour opérateur deux barres droites verticales (||).

```
DCL ZA CHAR(100) VAR;
DCL ZB CHAR(7);
DCL ZC CHAR(2);
DCL ZD CHAR(13);
ZB = 'EXEMPLE';
ZC = 'DE';
ZD = 'CONCATENATION';
ZA = ZB ||ZC ||ZD;
```

Cette dernière instruction signifie que *ZA* contiendra les informations qui se trouvent dans *ZB,* dans *ZC* et dans *ZD*, dès lors *ZA* aura comme contenu : EXEMPLEDECONCATENATION.

L'absence de blanc entre les mots est normale puisque l'opération de concaténation relie entre eux les différents contenus des zones. Pour obtenir des blancs, il faut écrire : *ZA = ZB || 'b' ||ZC || 'b' ||ZD;* à ce moment *ZA* contiendra : EXEMPLE DE CONCATENATION.

```
DCL ZA CHAR(100) VAR;
DCL ZB CHAR(20) VAR;
DCL ZC CHAR(20) VAR;
ZA = 'EXEMPLEb';
ZB = 'DEb';
ZC = 'CONCATENATION';
ZA = ZA || ZB || ZC;
```

L'opération *ZA = 'EXEMPLEb'*: fait de *ZA*, zone variable, une zone de huit positions qui contient le mot EXEMPLE suivi d'un blanc. La zone *ZB* contient le mot DE et un blanc, enfin *ZD* contient le mot CONCATENATION. La dernière instruction attribue à *ZA* le contenu EXEMPLE DE CONCATENATION. Si l'on avait écrit *ZA = ZA || ZA || ZA;* le contenu de *ZA* aurait été : EXEMPLE EXEMPLE EXEMPLE.

La longueur potentielle de la zone *ZA* doit être au moins égale à la somme des longueurs réelles des zones concaténées, autrement il y a perte d'informations.

La concaténation de zones de longueur fixe n'est guère utile car elle risque de créer des contenus qui ne répondent pas au besoin du traitement :

```
DCL ZL CHAR(100);
DCL ZA CHAR(20) INIT('EXEMPLE');
DCL ZB CHAR(20) INIT('DE');
DCL ZC CHAR(20) INIT('CONCATENATION');
ZL = ZA || ZB || ZC;
```

Cette opération donne à *ZL* le contenu :
*EXEMPLE*bbbbbbbbbbbbb*DE*bbbbbbbbbbbbbbbbb*CONCATENATION*bbbbbbb
L'ordinateur a bien transporté les blancs qui suivent les mots dans les trois zones.

17. La fonction translate.

Avant de faire quelques exercices, je voudrais encore signaler une fonction fort utile pour le traitement des données alphanumériques : la fonction *TRANSLATE* qui permet de remplacer dans une zone déterminée certains caractères par d'autres. Cette fonction a trois arguments :
1) le nom de la zone dans laquelle on doit remplacer des caractères,
2) les caractères qui doivent servir à en remplacer d'autres,

3) les caractères à remplacer.

```
DCL ZA CHAR(10) INIT('RADEAU');
DCL ZB CHAR(30);
ZB = TRANSLATE(ZA,'C','R');
```

Cette instruction signifie :

REMPLACER DANS ZA LA LETTRE 'R' PAR LA LETTRE 'C'.

Par conséquent, *ZB* contiendra le mot CADEAU.

```
ZB = TRANSLATE(ZA,'CV','RD');
```

Dans ce cas la lettre *'R'* sera remplacée par *'C'* et la lettre *'D'* par la lettre *'V'*; dès lors, *ZB* contiendra le mot *'CAVEAU'*.

```
DCL ZA CHAR(10) INIT('RADEAU');
DCL ZB CHAR(10);
DCL ZC CHAR(2) INIT('CV');
DCL ZD CHAR(2) INIT('RD');
ZB = TRANSLATE(ZA,ZC,ZD);
```

Cette instruction aura le même effet que la précédente; la zone *ZB* contiendra 'CAVEAU'. En fait, dans ce cas, l'ordinateur remplace dans *ZA* le contenu de *ZD* par celui de *ZC*.

On aurait pu écrire également :

```
ZB = TRANSLATE(ZA,'CV',ZD);
ZB = TRANSLATE(ZA,ZC,'RD');
ZA = TRANSLATE(ZA,'CV','RD');
```

Dans ce dernier cas, il n'y a pas de transmission de l'information dans une autre zone : il y a simplement substitution des caractères *R* par *C* et *D* par *V* dans la zone *ZA*.

EXERCICES

La meilleure manière d'assimiler une langue consiste à rendre aussi vivantes que possible les connaissances théoriques acquises en les utilisant dans la pratique. Il en va de même pour les langages de programmation. Etudier le PL/1 n'est possible que si les notions théoriques sont employées dans des problèmes concrets. Je voudrais, dès lors, dans les pages qui suivent, proposer quelques exercices qui permettront d'acquérir les automatismes indispensables dans la pratique courante d'un langage.

Dans une première série d'exercices, nous traiterons un fichier de cartes mécanographiques qui contient un texte perforé en continu dans les colonnes 1 à 75; les mots sont séparés les uns des autres par un blanc typographique, par l'apostrophe ou par une des ponctuations suivantes : , ; : . ? !

Par ailleurs, on notera qu'un mot ne peut pas être coupé; il se termine obligatoirement sur la carte où il commence.

Echantillon du fichier d'entrée

MESSIEURS LES BEAUX ESPRITS, DONT LA PROSE ET LES VERS SONT D'UN STYLE POMPEUX ET TOUJOURS ADMIRABLE, MAIS QUE L'ON N'ENTEND POINT, ECOUTEZ CETTE FABLE, ET TACHEZ DE DEVENIR CLAIRS.

FLORIAN, Le Singe qui montre la lanterne magique.

Exercice n°1

Ce premier exercice consiste à répartir les mots en fonction de leur lettre initiale. Le but est d'obtenir sur un listing la fréquence de chacune des lettres de l'alphabet en début de mot.

Le programme comprendra 4 étapes :

1) Lecture d'une carte dans la zone *ZL;*
2) Comptage de la lettre initiale du premier mot de la zone.
3) Déplacement du contenu de *ZL* de la droite vers la gauche de manière à éliminer le premier mot de la zone. Ce procédé permettra de placer en début de *ZL* chacun des mots contenus dans la carte. Lorsque la zone *ZL* contiendra uniquement des blancs, le traitement d'une carte sera terminé.
4) Impression des résultats.

Pour faire glisser le contenu de *ZL* de la droite vers la gauche, il faut repérer la première lettre d'un mot; celle-ci se trouve toujours dans la position qui suit soit un blanc soit une apostrophe, de sorte qu'il sera nécessaire de rechercher ces signes dans *ZL*. Pour éviter la recherche de ces deux signes, on substituera un blanc à l'apostrophe de sorte que l'on pourra considérer qu'un mot commence à la position qui suit un blanc.

Les déclarations nécessaires pour ce problème sont :

1) Une zone de lecture (*ZL*) de 80 positions alphanumériques.
2) Une table de 26 éléments alphanumériques d'une position dans laquelle on rangera l'alphabet (*TA*).
3) Une table de 26 compteurs de 5 positions pour dénombrer la fréquence à l'initiale de chacune des lettres (*TB*).

```
1  PRO1 : PROC OPTIONS(MAIN);
2  DCL ZL CHAR(80);
3  DCL IN FILE RECORD INPUT;
4  DCL TA(26) CHAR(1) INIT ('A','B','C','D','E','F','G','H','I',
5                           'J','K','L','M','N','O','P','Q','R',
6                           'S','T','U','V','W','X','Y','Z');
7  ON ENDFILE(IN) GOTO FIN;
8  DCL TB(26) FIXED DEC(5) INIT((26)0);
9  AA : READ FILE(IN) INTO(ZL);
10      ZL=TRANSLATE (ZL,'ƀ','''');
11 BB : DO J = 1 TO 26;
12              IF SUBSTR(ZL,1,1) = TA(J) THEN DO;
13                      TB(J) = TB(J) + 1;
14                      GOTO AB;
15                      END;
16      END;
17 AB : L = INDEX(ZL,'ƀ');
18         SUBSTR(ZL,1) = SUBSTR(ZL,L + 1);
19         IF ZL = 'ƀ' THEN GOTO AA;
20         GOTO BB;
21 FIN : DO J = 1 TO 26;
22              PUT EDIT(TA(J), TB(J)) (SKIP,A(1), X(2), F(5));
```

```
23          END;
24          END;
```

Commentaires

L'instruction 10 permet de remplacer l'apostrophe par un blanc (représenté par b). On remarquera que le troisième argument de la fonction *TRANSLATE* se compose d'une suite de quatre apostrophes. En voici la raison. Le premier et le dernier signes servent à délimiter une constante alphanumérique; nous connaissons bien cet emploi. Mais cet usage pose un problème lorsque le signe à placer entre apostrophes est lui-même une apostrophe. En PL/1, on doit répéter l'apostrophe une deuxième fois, c'est ce qui explique la quadruple apparition de ce signe dans le troisième argument de la fonction *TRANSLATE*.

L'opération 12 a pour but de comparer chacun des éléments de la table *TA* à la première position de la zone *ZL (SUBSTR(ZL,1,1))*. Lorsqu'il y a identité entre un élément de *TA* et la première position de *ZL*, l'ordinateur ajoute 1 à l'élément compteur de *TB* qui correspond à l'élément de *TA*.

Les instructions 17 et 18 permettent de trouver la position de la première lettre du mot qui doit être traité et de déplacer le contenu de *ZL* de la droite vers la gauche. Voici pour la première carte du fichier les valeurs de l'indice *L* et le contenu de *ZL* après chaque déplacement.

ZL contient au départ après l'opération de *TRANSLATE* :
MESSIEURS LES BEAUX ESPRITS, DONT LA PROSE ET LES VERS

L contenu de *ZL* après déplacement.

```
10 LES BEAUX ESPRITS, DONT LA PROSE ET LES VERS
 4 BEAUX ESPRITS, DONT LA PROSE ET LES VERS
 6 ESPRITS, DONT LA PROSE ET LES VERS
 9 DONT LA PROSE ET LES VERS
 5 LA PROSE ET LES VERS
 3 PROSE ET LES VERS
 6 ET LES VERS
 3 LES VERS
 4 VERS
 5 bbbbbbbbbbbbbbbbbbbbbbb
```

On peut voir grâce à cet exemple que la valeur de *L* donne la position du premier blanc dans la zone *ZL*. Cette position précède immédiatement la première lettre du mot suivant de sorte que le déplacement sera *SUBSTR(ZL,L + 1)*.

La fin du programme se comprend sans peine : l'ordinateur imprime les 26 lettres de l'alphabet avec, en regard de chacune d'elles, le nombre de fois qu'elles apparaissent en début de mot.

Exercice n°2

On doit calculer la fréquence d'apparition de chaque mot dans un fichier. Comme le texte est enregistré sur cartes mécanographiques dans les colonnes 1 à 75, le travail se déroulera en trois étapes.

1) La première étape consistera à constituer un fichier sur disques magnétiques dans lequel chaque mot sera isolé des autres et placé dans une zone de 80 positions.
2) Les mots seront triés en ordre alphabétique de manière à regrouper dans le fichier les formes identiques. Le tri pourra se faire en utilisant un programme fourni par le constructeur de l'ordinateur. Le programmeur doit simplement donner à la machine des indications sur la longueur et la nature des données à trier. Etant donné que les programmes de tri doivent être présentés sous une forme qui répond aux caractéristiques de l'ordinateur que l'on utilise, il faut s'adresser aux responsables du Centre de Calcul où l'on travaille pour obtenir un programme de tri standard.
3) La troisième étape du traitement effectuera le comptage proprement dit de la fréquence des formes puisque les deux phases précédentes ne sont en fait que des tâches préparatoires. Nous avons déjà réalisé un programme de calcul de fréquence au début de ce chapitre consacré au PL/1 (voir le problème n°2 solution 3).

Le seul aspect du travail dont nous devons nous occuper est donc le premier : constituer un programme qui lira les cartes, isolera les mots les uns des autres et les placera un par un sur disques magnétiques en utilisant une zone d'écriture de 80 positions alphanumériques. Deux zones de travail seront employées : une zone de 80 positions alphanumériques *ZL* et une zone d'écriture *YL* de 80 positions alphanumériques également. Le programme comprendra trois étapes :

1) Lecture des cartes dans *ZL*.
2) Transmission dans *YL* de chacun des mots contenus dans *ZL;* cette transmission se fait mot par mot et, dès qu'un mot est transmis dans *YL*, l'ordinateur effectue

la phase trois.

3) Ecriture du contenu de *YL* sur les disques magnétiques.

```
1  EX : PROC OPTIONS(MAIN);
2  DCL ZL CHAR(80);
3  DCL YL CHAR(80);
4  DCL IN FILE RECORD INPUT;
5  DCL OUT FILE RECORD OUTPUT;
6  ON ENDFILE(IN) GOTO FIN;
7  AA : READ FILE(IN) INTO(ZL);
8         YL = 'ƀ';
9         K = 0;
10        ZL = TRANSLATE(ZL,'ƀƀƀƀƀƀƀ','"„::.? ! ');
11        DO L = 1 TO 75;
12        IF SUBSTR(ZL,L,1) = 'ƀ' THEN DO;
13        IF YL ¬= 'ƀ' THEN WRITE FILE(OUT) FROM (YL);
14                          YL = 'ƀ';
15                          K = 0;
16                          GOTO AB;
17                          END;
18        K = K + 1;
19        SUBSTR (YL,K,1) = SUBSTR(ZL,L,1);
20 AB : END;
21        GOTO AA;
22 FIN : CLOSE FILE(OUT);
23         END;
```

Commentaires

L'indice *L* que l'on emploie dans les opérations 11, 12 et 19 sert à désigner la position de *ZL* sur laquelle on travaille tandis que l'indice *K* des instructions 9, 15, 18 et 19 désigne la position de *YL*. Ces deux indices permettent de progresser position par position l'un dans le balayage de la zone *ZL*, l'autre dans la transmission progressive des caractères dans la zone *YL*.

Les opérations des lignes 13 à 17 ne sont exécutées que lorsque l'ordinateur a repéré

une position blanche dans *ZL*, c'est-à-dire lorsqu'un mot entier a été transmis dans *YL*. En effet, c'est à ce moment seulement que l'on doit écrire sur disques le contenu de *YL*. L'instruction 13 commence par vérifier si la zone *YL* contient autre chose que des blancs. La raison de ce test est d'éviter d'inscrire sur disques une zone qui ne contient rien. A quel moment l'ordinateur risque-t-il de transmettre sur disques une zone blanche? Reprenons la première ligne du texte de *FLORIAN.*

```
          b   b       b       b     b  b      b  b   b      bbbb...
MESSIEURS LES BEAUX ESPRITS, DONT LA PROSE ET LES VERS
```

L'ordre de *TRANSLATE* en 10 qui consiste à remplacer les ponctuations et l'apostrophe par des blancs donne à *ZL* la valeur suivante :

```
          b   b       b       bb    b  b      b  b   b      bbbb...
MESSIEURS LES BEAUX ESPRITS  DONT LA PROSE ET LES VERS
```

L'ordinateur commence ensuite l'exécution du *DO* itératif de la ligne 11. Voici les valeurs de *L* et de *K* au moment où l'ordinateur détecte un blanc dans *ZL;* j'ai ajouté au tableau le contenu de *YL* :

L	*K*	*YL*
10	9	MESSIEURS
14	3	LES
20	5	BEAUX
28	7	ESPRITS
29	0	bbbbbbbbbb.....
34	4	DONT

. . . . . .

On peut voir que le contenu de *YL*, au moment où *L* vaut 29 est nul. La raison en est que, à la suite du mot ESPRITS, il y a dans *ZL* deux blancs consécutifs : lorsque *L* est égal à 28, l'ordinateur détecte un blanc dans la position 28 de *ZL*, il écrit sur disques le contenu de *YL* (ESPRITS) et remet cette zone à blanc. Ensuite, l'ordinateur continue l'exécution du *DO L = 1 TO 75* en attribuant à l'indice *L* la valeur 29, comme la vingt-neuvième position de *ZL* est blanche, l'ordinateur va de nouveau à l'instruction 13 bien que *YL* ne contienne aucune information.

L'instruction 22 -*CLOSE FILE(OUT);*- n'a jamais été rencontrée jusqu'à présent. Elle signifie que l'ordinateur doit *clôturer le fichier OUT* que l'on a enregistré sur disques magnétiques. Comme on peut le constater, cette instruction se compose du mot *FILE* avec entre parenthèses le nom du fichier à clôturer.

La fermeture est obligatoire pour tout *fichier de sortie* que l'on enregistre sur bandes ou sur disques magnétiques afin que l'ordinateur marque d'un signe spécial la fin du fichier.

On remarquera que cette opération s'exécute *une seule fois* au moment où toutes les données ont été traitées et écrites sur les disques magnétiques.

La solution que j'ai proposée pour ce deuxième exercice conduit l'ordinateur à parcourir la zone *ZL* position par position et à transmettre dans *YL* un seul caractère à la fois. Il est possible de rédiger le programme d'une autre manière en utilisant les fonctions *INDEX* et *SUBSTR.*

```
1  EX : PROC OPTIONS(MAIN);
2  DCL ZL CHAR(80);
3  DCL YL CHAR(80) INIT('b');
4  DCL IN FILE RECORD INPUT;
5  DCL OUT FILE RECORD OUTPUT;
6  ON ENDFILE(IN) GOTO FIN;
7  AA : READ FILE(IN) INTO(ZL);
8       ZL = TRANSLATE(ZL,'bbbbbbb','",:;.? ! ');
9       J = 1;
10 AB : IF SUBSTR(ZL,J,4) = 'bbbb' THEN GOTO AA;
11      L = INDEX(SUBSTR(ZL,J),'b');
12      YL = SUBSTR(ZL,J,L-1);
13      J = J + L;
14      IF YL ¬= 'b' THEN WRITE FILE(OUT) FROM(YL);
15      GOTO AB;
16 FIN : CLOSE FILE(OUT);
17      END;
```

Commentaires

En principe, ce programme doit se comprendre sans difficulté : les notions théoriques sur lesquelles il se fonde ont été longuement exposées dans les paragraphes consacrés aux fonctions *INDEX* et *SUBSTR* et à leurs emplois combinés.

Seule, peut-être, l'instruction 10 demande un mot d'explication. Il s'agit d'une interrogation qui vérifie si la zone *ZL*, à partir de la position *J*, contient quatre blancs consécutifs. L'indice *J*, comme on peut le constater, désigne successivement les différentes positions de *ZL* qui suivent un blanc. Lorsque l'ordinateur a parcouru toute la partie occupée de *ZL*, il doit retourner à la lecture pour prendre la carte suivante du fichier. Or le moyen de voir si tous les mots contenus dans *ZL* ont été traités est de contrôler si, à partir de l'endroit où l'on se trouve dans *ZL*, il y a au moins quatre blancs qui se succèdent.

La remise à blanc de *YL* n'est pas nécessaire dans cette solution car les informations sont transmises en bloc dans cette zone de sorte que la partie inoccupée de *YL* est automatiquement remise à blanc en raison des règles propres à l'instruction de transmission.

Exercice n°3

Partant du fichier que l'on vient d'enregistrer sur disques au cours de l'exercice précédent, on doit constituer une concordance des diverses occurrences du mot DONT en imprimant le mot entre deux astérisques et en le faisant précéder et suivre d'un contexte de quatre mots. On doit donc obtenir le résultat suivant :

MESSIEURS LES BEAUX ESPRITS*DONT*LA PROSE ET LES

On peut imaginer plusieurs méthodes pour réaliser un tel programme. Celle qui me semble la plus directe consiste à utiliser comme zone de stockage d'un contexte de 9 mots, une table composée de 9 éléments alphanumériques de longueur variable, soit *TA(9) CHAR(50) VAR;* On imprimera le contenu de la table lorsque le cinquième élément contiendra le mot DONT.

La technique de recherche du mot nécessite que l'on range successivement tous les mots du texte dans chacun des éléments de la table de la façon suivante :

1) Lecture d'un mot dans la zone *ZL*.
2) Transmission de *ZL* dans le neuvième élément de *TA*.

3) Vérification du contenu de l'élément 5 de *TA;* si ce dernier contient le mot DONT, impression de tous les éléments de *TA;* autrement passage immédiat à la quatrième opération.
4) Transmission de l'élément n de *TA* dans l'élément n-1; ainsi, le contenu du deuxième élément passera dans le premier, celui du troisième dans le deuxième et ... celui du neuvième dans le huitième. Ensuite, retour à la lecture pour aller chercher le mot suivant qui sera rangé dans *TA(9).*

Au début du programme, l'ordinateur va vérifier le contenu de *TA(5)* bien que cet élément ne contienne aucune information. En effet, puisque les mots sont rangés dans *TA(9)* et qu'ils remontent de proche en proche vers *TA(1), TA(5)* contiendra une donnée après la cinquième lecture seulement. Cela n'a aucune importance puisque ce que l'on recherche, c'est la présence du mot DONT dans *TA(5);* le fait que cet élément ne contienne rien n'entrave pas le travail de l'ordinateur.

Au moment où l'ordinateur détecte la fin du fichier, les éléments 1 à 8 de *TA* contiennent les huit derniers mots du texte et les éléments 5, 6, 7 et 8 qui n'ont pas encore été contrôlés par l'ordinateur, pourraient théoriquement contenir une occurrence du mot recherché. Il faut dès lors continuer le processus décrit aux points 3 et 4 ci-dessus mais sans effectuer de lecture. Le traitement ne pourra être arrêté que si *TA(9)* et *TA(1)* contiennent simultanément des blancs.

Nous rencontrerons dans ce programme une notion nouvelle de PL/1 et un emploi particulier du format *A* dans un *PUT EDIT.*

1.- Le OU logique.

On peut, dans un *IF*, exprimer successivement plusieurs conditions et exécuter la partie *THEN* de l'instruction si l'une des conditions est réalisée; cela revient à dire : Si telle condition est réalisée ou telle autre ou ... *ALORS*

En PL/1, on représente le *OU* par une barre verticale | (ne pas confondre avec la concaténation qui se représente par deux barres verticales).

IF J = 5 | J = 6 THEN ....
Si *J* vaut 5 ou 6, alors on exécute ce qui suit *THEN ....*

A côté du *OU* logique, on peut rencontrer en PL/1, le *ET* qui se présente par & (le *et* commercial).

IF J = 5 & L = 6 THEN ....

L'expression qui suit *THEN* ne sera exécutée que si *J* vaut 5 et que si, en même temps, *L* vaut 6.

On notera qu'une instruction telle que : *IF J = 5 & J = 6 THEN ....* n'a aucun sens car l'indice *J* ne peut prendre simultanément les valeurs 5 et 6.

2.- Dans l'instruction *PUT EDIT(ZL) (SKIP,A(80));*, la mention (80) est facultative pour les zones alphanumériques; dès lors, on peut écrire : *PUT EDIT(ZL) (SKIP,A);* Lorsque la longueur de la zone n'est pas indiquée, l'ordinateur la calcule. Cette facilité est utile lorsqu'on imprime des zones déclarées *VARYING* : le programmeur n'a pas à se soucier de la longueur effective de la zone à imprimer.

Nous pouvons à présent écrire le programme de concordance du mot DONT. La zone de lecture des données s'appelle *ZL* et la table de stockage des mots *TA*.

```
1  EX : PROC OPTIONS(MAIN);
2  DCL ZL CHAR(80);
3  DCL TA(9) CHAR(50) VAR;
4  DCL IN FILE RECORD INPUT;
5  ON ENDFILE(IN) GOTO FIN;
6  TA = 'b';
7  AA : J = 9;
8        READ FILE(IN) INTO(ZL);
9        TA(J) = SUBSTR(ZL,1, INDEX(ZL,'b'));
10 AB : IF TA(5) ¬= 'DONTb' THEN GOTO CA;
11       PUT SKIP;
12       DO L = 1 TO 9;
13            PUT EDIT(TA(L)) (A);
14            IF L = 4 | L = 5 THEN PUT EDIT('*') (A);
15       END;
16 CA : DO J = 2 TO 9;
17       TA(J - 1) = TA(J);
18       END;
19       IF TA(9) = 'b' THEN IF TA(1) = 'b' THEN GOTO FINI;
20                                          ELSE GOTO AB;
```

```
21          GOTO AA;
22  FIN : TA(9) = 'b';
23          GOTO AB;
24  FINI : END;
```

Commentaires

L'instruction 6 *TA* = 'b'; a pour effet de transmettre un blanc dans chacun des éléments de *TA*. Comme cette table a des éléments de longueur variable, chacun d'eux occupera une position dans l'unité centrale.

L'opération 9 fait passer dans *TA(J)* tout ce qui se trouve dans *ZL* jusqu'à la première position blanche. On notera que le blanc qui suit immédiatement le mot contenu dans *ZL* est également transporté dans *TA*. On en verra la raison à la ligne 13.

Le groupe d'instructions 11 à 15 n'est exécuté que si le cinquième élément de *TA* contient le mot DONT. En 11, l'ordinateur provoque un saut de papier à l'imprimante de sorte que l'on peut commencer l'impression d'une nouvelle ligne. L'opération 13 sert à imprimer l'élément de *TA* désigné par *J*. On observe que l'ordinateur imprime le contenu de *TA(J)* sans faire de saut à l'imprimante de sorte que les 9 éléments se trouveront sur la même ligne, les uns à la suite des autres sans autre blanc typographique que celui que l'on a transmis dans chacun des éléments de *TA* en même temps que le mot (cf. instruction 9). L'opération 14 a pour but de placer un astérisque avant et après le mot DONT. Le *PUT EDIT* sera exécuté soit quand *L* vaut 4 (avant le mot) soit quand il vaut 5 (après le mot).

Les opérations 16 à 18 ont pour but de déplacer le contenu de chacun des éléments de *TA* dans celui qui le précède selon la technique que nous avons expliquée. On remarque que l'élément 1 de *TA* n'est transmis nulle part mais est remplacé par le contenu de l'élément 2.

Il reste à expliquer les instructions propres aux dernières données du fichier. Lorsque l'ordinateur détecte la fin du fichier, il passe à l'ordre 22 (étiquette *FIN*) où, au lieu de lire une nouvelle donnée, il transporte un blanc dans *TA(9);* ensuite, il vient à *AB.* La fin du traitement qui est postérieure à la détection de la fin du fichier est vérifiée par l'ordre 19. En fait, le programme se termine lorsque tous les éléments de *TA* contiennent un blanc , ce qui sera le cas si on a simultanément un blanc dans *TA(1)*

et dans *TA(9)*. L'instruction 19 doit être comprise comme suit :

SI *TA(9)* et *TA(1)* CONTIENNENT UN BLANC ARRET A FINI.
SI *TA(9)* CONTIENT UN BLANC MAIS PAS *TA(1)* RETOUR A *AB.*
SI *TA(9)* et *TA(1)* CONTIENNENT AUTRE CHOSE QU'UN BLANC, RETOUR A *AA.*

En réalité, *TA(1)* contiendra un blanc au début du programme avant que l'ordinateur n'ait lu 9 données. *TA(9)* contiendra un blanc à la fin du programme, après la lecture de la dernière donnée et durant le traitement des huit derniers mots du fichier.

*

* *

Nous avons terminé l'étude des principales instructions du langage PL/1. Je rappelle que je n'ai pas voulu donner une description complète du PL/1 mais seulement un aperçu général des possibilités qu'il offre pour le traitement des données alphanumériques. Pour certaines instructions, je me suis strictement limité à ce qui m'a semblé être le plus utile dans le but de ne pas alourdir exagérément cet exposé. Ainsi, par exemple, le *DO* itératif aurait demandé à être explicité davantage; de même, on aurait pu s'attarder quelque peu aux opérations logiques "et" et "ou". Je ne puis que conseiller à ceux qui veulent en savoir plus de se reporter aux ouvrages que je cite en annexe.

LA GESTION DES FICHIERS

Une des caractéristiques principales du traitement de l'Information en Sciences humaines est qu'il nécessite généralement l'exploitation de fichiers volumineux. Dès lors, un exposé sur les méthodes de programmation ne saurait être complet s'il ne comprenait pas un chapitre consacré à la gestion des fichiers.

1. QU'EST-CE QU'UN FICHIER ?

Un fichier est un ensemble de données qui présentent des points communs tant au point de vue de leur contenu qu'au point de vue de leur constitution. L'exemple le mieux connu de fichier est un ensemble de cartes mécanographiques dans lesquelles on a perforé les informations relatives à tel ou tel domaine de la recherche. Ainsi, le chercheur qui a effectué une enquête sociologique sur une population définie constituera un fichier de sa population en portant sur cartes les résultats de son enquête.

Nous verrons qu'en informatique, on appelle fichier toute collection de données qui se trouve sur un support compréhensible par l'ordinateur : cartes, bandes magnétiques, disques magnétiques, rubans perforés etc...

2. QU'EST-CE QUE GERER UN FICHIER ?

C'est le conserver sur un support que l'ordinateur peut lire, en assurant sa bonne conservation et sa mise à jour. C'est aussi choisir le support le mieux adapté au traitement que l'on doit exécuter en utilisant les méthodes d'accès aux données les plus adéquates.

3. COMMENT L'ORDINATEUR PEUT-IL ACCEDER AUX DIFFERENTES DONNEES D'UN FICHIER ?

La manière d'accéder aux différentes données concerne à la fois la lecture et l'écriture de celles-ci.

1) L'ordinateur peut accéder aux données de manière séquentielle, c'est-à-dire en suivant l'ordre où elles figurent dans le fichier.

Lorsqu'on déclare : *DCL FIC FILE RECORD INPUT;*, l'ordinateur interprète la définition et considère que l'accès aux données sera *séquentiel.*

Les fichiers de cartes mécanographiques et les fichiers enregistrés sur bandes magnétiques *sont obligatoirement des fichiers séquentiels.* On n'imagine pas, en effet, que

l'ordinateur lise la 9000ème carte d'un fichier avant d'avoir lu les 8999 cartes qui précèdent. De même, la bande magnétique qui est un ruban continu sur lequel les informations sont stockées les unes à la suite des autres, ne peut être traitée que séquentiellement.

2) Des données enregistrées sur disques magnétiques peuvent également être lues ou écrites séquentiellement. Mais en outre, le disque, de par sa structure, autorise des méthodes d'accès plus rapides. Nous étudierons l'une d'entre elles : *l'accès indexé-séquentiel aux informations.*

Pour définir cette méthode de consultation des fichiers, je me référerai à la manière dont on recherche un mot dans un dictionnaire.

Une première possibilité consiste à parcourir le dictionnaire mot par mot jusqu'à la découverte du mot recherché. Jamais personne ne procède ainsi, il faudrait trop de temps pour retrouver une information. Une façon plus directe de rechercher un mot consistera à travailler en deux étapes :

1) On consultera la table des matières pour voir à quelles pages se trouvent les mots qui commencent par la même lettre que le mot que l'on recherche. On ouvrira le volume à la page indiquée par la table.

2) Dans un dictionnaire, on peut lire, en haut de chaque page le premier et le dernier mots contenus dans cette page. En utilisant ces indications, on pourra parcourir assez rapidement le volume jusqu'à ce qu'on découvre au haut d'une page un mot qui, dans l'ordre alphabétique, suit le mot que l'on recherche. A ce moment, on devra lire tous les mots qui se trouvent sur la page jusqu'à celui qui nous intéresse.

Ce mode de consultation que l'on emploie pour rechercher un mot dans un dictionnaire est *indexé-séquentiel.* Il est *indexé* en ce sens qu'il n'oblige pas le lecteur à parcourir toutes les données de l'ouvrage puisqu'il utilise des points de repère qui accélèrent la consultation. Il est *séquentiel* puisque le lecteur doit lire tous les mots qui figurent sur la page où se trouve le mot.

Le langage PL/1 permet de constituer et de consulter des fichiers indexés-séquentiels. Il faut pour cela utiliser dans la déclaration du fichier un certain nombre de précisions que nous allons examiner en détail.

DCL IN FILE RECORD INPUT;

Cette déclaration que nous connaissons bien est destinée à définir un fichier séquentiel d'entrée (*INPUT*). L'addition du mot *KEYED* aux éléments qui précèdent indique que le fichier sera en outre accessible par des clés.

Une clé (KEY) est un élément qui permet d'identifier une donnée, de la distinguer des autres et de la retrouver dans un fichier. Reprenons l'exemple du dictionnaire. La clé sera le *mot-rubrique* que l'on cherche. C'est à partir de ce mot-rubrique que l'on trouvera la définition du mot.

Il existe d'autres similitudes entre un dictionnaire et un fichier indexé-séquentiel :

1) Dans un dictionnaire, les mots doivent être rangés en ordre alphabétique strict autrement le lecteur risque de ne pas retrouver certains mots. De même, dans un fichier indexé-séquentiel, les clés doivent être rangées en ordre croissant (ordre alphabétique ou classement numérique).

2) Dans un dictionnaire, il ne doit pas y avoir deux mots-rubriques identiques (cela se produit pour les homographes, mais généralement ils sont distingués par un chiffre ou par une lettre); ainsi, dans un fichier indexé-séquentiel, une clé doit être unique, elle ne peut figurer qu'une fois dans le fichier.

Exemples de fichiers indexés-séquentiels.

1) On doit enregistrer sur disques magnétiques un fichier qui contient la liste des communes d'un pays et leur numéro postal. Ce numéro servira de clé :

1000	BRUXELLES
1920	DIEGEM
2000	ANVERS
3500	HASSELT
4000	LIEGE
5000	NAMUR
6000	CHARLEROI
7000	MONS
8000	BRUGES
9000	GAND

Ce fichier peut être indexé-séquentiel si le numéro postal sert de clé, en effet :

- chaque numéro ne figure qu'*une seule fois* dans le fichier;

- les numéros sont rangés en ordre croissant.

Supposons à présent que le nom de la localité doive servir de clé, le fichier se présentera comme suit :

2000	ANVERS
8000	BRUGES
1000	BRUXELLES
6000	CHARLEROI
1920	DIEGEM
9000	GAND
3500	HASSELT
4000	LIEGE
7000	MONS
5000	NAMUR

Dans ce cas ce sont les noms des localités qui sont classés en ordre alphabétique croissant.

2) Un autre exemple de fichier indexé-séquentiel pourrait être un lexique dans lequel les mots sont accompagnés d'une brève définition. Les mots qui sont rangés en ordre alphabétique, pourront servir de clé et permettre de retrouver automatiquement les définitions.

Exemple

âge	durée de la vie
agencer	ajuster, arranger
aiguiser	rendre aigu
air 1	fluide gazeux qui forme l'atmosphère
air 2	manière, façon
air 3	suite de notes composant un chant
aise	état agréable
alerter	avertir qu'il y a danger

Dans ce cas, le mot sert de clé; dès lors, les enregistrements qui composent le fichier sont classés selon l'ordre alphabétique des mots. On remarque que pour les homographes (air), on a fait suivre le mot d'un chiffre qui correspond à sa place dans le fichier.

La raison en est que chaque clé doit être unique et permettre d'accéder à un seul enregistrement.

4. L'ATTRIBUT ENVIRONMENT DANS LA DECLARATION D'UN FICHIER.

Parmi les attributs possibles dans une déclaration de fichier, il en est un dont nous n'avons pas encore parlé, c'est l'attribut *ENVIRONMENT* dont l'abréviation est *ENV.* Le mot annonce une série de précisions relatives à la constitution physique du fichier. Dans une déclaration, le mot *ENVIRONMENT* est suivi des indications nécessaires à la définition du fichier; ces indications sont placées entre parenthèses.

DCL OUT FILE RECORD OUTPUT ENV(.....);

Les indications que l'on peut trouver dans l'*ENVIRONMENT* d'un fichier sont nombreuses, je me limiterai aux principales :

1) *RECSIZE* (nombre d'octets de l'enregistrement).
Le mot *RECSIZE* signifie "taille du record", c'est-à-dire longueur en octets de l'enregistrement. Pour une carte mécanographique, par exemple, la longueur sera 80 soit *RECSIZE(80).*
2) *BLKSIZE* (nombre de positions d'un groupe d'enregistrements).
Lorsqu'on enregistre un fichier sur disques ou sur bandes magnétiques, il est souhaitable de constituer des *"BLOCS"*, c'est-à-dire d'effectuer un regroupement des données sur les disques ou sur les bandes; en voici la raison : l'ordinateur, lorsqu'il effectue un ordre d'écriture, laisse entre le dernier enregistrement qu'il a stocké et celui qu'il va écrire un espace appelé "gap" ce qui entraîne une perte de place considérable sur le support. Ainsi, si l'on veut enregistrer dix données de 5 positions sur une bande avec un fichier déclaré *RECSIZE(5)* et sans précision de *BLKSIZE*, la bande se présentera comme suit :

Enr. 1 GAP Enr. 2 GAP Enr. 3 GAP Enr. 4 GAP ..... Enr. 10 ....

On voit le gaspillage de place que ce système provoque. Par contre l'option *BLKSIZE* permet de diminuer la perte de place et, par conséquent, le nombre de *GAPS* car l'ordinateur effectue moins d'opérations d'écriture. Le nombre que l'on indique entre parenthèses à la suite du mot *BLKSIZE* doit être un multiple du nombre indiqué à la suite de *RECSIZE.* Ainsi, dans le cas de notre fichier de 10 données de 5 positions, on pourra

écrire *RECSIZE(5) BLKSIZE(25)*, ce qui donnera à la bande la configuration suivante :

Enr. 1 Enr. 2 Enr. 3 Enr. 4 Enr. 5 GAP Enr. 6 Enr. 7 .....

Le gain d'espace sur le support magnétique est évident puisqu'il n'y aura qu'un gap au lieu de 9. Si l'on avait écrit *RECSIZE(5) BLKSIZE(50)*, il n'y aurait pas eu de gap sur le support. Le choix du nombre à la suite de *BLKSIZE* doit se faire en fonction des types de disques et de bandes que l'on utilise et également en fonction de la place dont on dispose dans l'unité centrale de traitement.

3) Aux indications de longueur de l'enregistrement et de longueur du groupe, il faut ajouter une précision sur leur nature : un enregistrement peut être de longueur fixe ou de longueur variable; dans le premier cas, on inscrira la lettre *F(FIXED)* dans l'*ENVIRONMENT,* dans le second cas, ce sera la lettre *V(VARIABLE).*
Par ailleurs, si l'on veut constituer des groupes d'enregistrement, on fera suivre immédiatement *F* ou *V* de la lettre *B(BLOCKED)*, on aura donc soit *FB*, soit *VB.*
Si l'on reprend l'exemple précédent, la déclaration complète sera :

DCL OUT FILE RECORD OUTPUT ENV(FB RECSIZE(5) BLKSIZE(25));

Ces trois indications que nous venons de voir concernent aussi bien les fichiers séquentiels que les fichiers indexés-séquentiels. Elles sont nécessaires *uniquement pour les fichiers de sortie(OUTPUT)* et elles sont enregistrées sur le support avant le début du fichier [1].

Les indications qui suivent sont propres aux fichiers indexés-séquentiels :

4) *INDEXED* indique que le fichier est indexé.

5) *KEYLENGTH* (longueur de la clé) précise la longueur de la clé en nombre d'octets. Dans le premier exemple de fichier indexé-séquentiel que nous avons vu, la clé était un numéro de quatre chiffres, on dira donc *KEYLENGTH(4).*

6) *KEYLOC* (numéro de la position où commence la clé) : ce mot sert à indiquer la première position de la clé dans l'enregistrement. On notera que la clé doit de préférence se trouver ailleurs qu'à la première position de l'enregistrement, dès lors *KEYLOC(1) est vivement déconseillé, voire même dans certains cas interdit.*

(1) L'écriture sur le support des précisions de l'*ENVIRONMENT* est entièrement automatique. Le programmeur ne doit pas s'en occuper.

Ces trois précisions sont nécessaires pour la déclaration des fichiers de sortie *(OUTPUT);* elles sont également enregistrées sur le support en même temps que la longueur de l'enregistrement et les autres indications de l'*ENVIRONMENT.*

Un fichier indexé-séquentiel doit être créé avec les informations suivantes :
1) le numéro postal;
2) la localité.
Le nom de la localité servira de clé, il commence à la position 5 de l'enregistrement et comprend au maximum 35 positions. La déclaration sera donc :

DCL OUT FILE RECORD OUTPUT KEYED ENV(INDEXED FB RECSIZE(39) BLKSIZE(3900) KEYLOC(5) KEYLENGTH(35));

5. LES INSTRUCTIONS PROPRES AUX FICHIERS INDEXES-SEQUENTIELS.

Pour décrire les instructions propres à ce type de fichiers, nous allons rédiger trois exercices, l'un consiste à créer un fichier, le deuxième à l'exploiter et le troisième à le mettre à jour.

A. La création d'un fichier.

Le fichier contient les informations que nous avons vues au paragraphe précédent, soit un numéro postal et un nom de localité par enregistrement.
Exemple : 4000LIEGE

L'instruction d'écriture pour un fichier indexé-séquentiel comprend outre l'expression *WRITE FILE* (nom du fichier) *FROM* (nom de la zone); le mot *KEYFROM* (nom de la zone où se trouve la clé). Ainsi, l'instruction sera :

WRITE FILE(OUT) FROM(ZL) KEYFROM(ZB);

OUT est le nom du fichier d'*OUTPUT;*

ZL est le nom de la zone d'écriture;

ZB est le nom de la zone où se trouve la clé.

```
1  EX : PROC OPTIONS(MAIN);
2  DCL IN FILE RECORD INPUT;
3  DCL OUT FILE RECORD OUTPUT ENV(FB RECSIZE(39) BLKSIZE(3900)
```

```
4      INDEXED KEYLOC(5) KEYLENGTH(35)) KEYED;
5   ON ENDFILE(IN) GOTO FIN;
6   DCL 1 ZL,
7           2 ZA CHAR(4),
8           2 ZB CHAR(35);
9   AA : READ FILE(IN) INTO(ZL);
10        WRITE FILE(OUT) FROM(ZL) KEYFROM(ZB);
11        GOTO AA;
12  FIN : CLOSE FILE(OUT);
13          END;
```

Commentaires

L'enregistrement a une longueur de 39 positions. Un groupe d'enregistrements comprendra 3900 divisé par 39, soit 100 données. La zone d'écriture *ZL* a au total 39 positions puisqu'elle doit correspondre à la longueur déclarée dans l'*ENVIRONMENT* du fichier *OUT*. Dans la déclaration de ce dernier, on remarque que le mot *KEYED* est placé en dernier lieu. En réalité, seule la place des mots est importante pour le début de la déclaration *DCL OUT FILE* pour le reste l'ordre des attributs est indifférent.

B. Exploitation du fichier.

Le fichier est à présent enregistré sur un disque magnétique, on doit l'utiliser pour traiter des données qui contiennent l'identité d'un certain nombre de personnes ainsi que leur adresse à l'exception du numéro postal. On désire par conséquent retrouver ce numéro. Pour chaque individu, on possède une carte qui contient les éléments suivant : col. 1 à 40 le nom et le prénom;
col. 41 à 80 l'adresse.
Dans l'adresse, le nom de la localité vient en premier lieu et il est immédiatement suivi d'un blanc.

Les opérations à effectuer seront :

1) Lecture d'une carte dans la zone *ZL;*
2) Recherche du nom de la localité en isolant les informations qui commencent à la colonne 41 et qui vont jusqu'au premier blanc qui suit cette colonne;
3) Consultation du fichier indexé-séquentiel pour y trouver le numéro postal. La clé

sera le nom de la localité.
4) Impression de chaque carte et du numéro postal.

L'instruction qui permet de lire une donnée dans un fichier indexé-séquentiel est *READ FILE*(nom du fichier) *INTO*(nom de la zone de lecture) *KEY*(nom de la zone où est la clé). Dans le programme, le nom du fichier indexé-séquentiel est *IND.*

```
1  EX : PROC OPTIONS(MAIN);
2  DCL IN FILE RECORD INPUT;
3  DCL IND FILE RECORD INPUT KEYED ENV(INDEXED);
4  ON ENDFILE(IN) GOTO FIN;
5  DCL 1 ZL,
6          2 ZA CHAR(40),
7          2 ZB CHAR(40);
8  DCL 1 XL,
9          2 XA CHAR(4),
10         2 XB CHAR(35);
11 AA : READ FILE(IN) INTO(ZL);
12       L = INDEX(ZB,'b');
13       XB = SUBSTR(ZB,1,L-1);
14       READ FILE(IND) INTO(XL) KEY(XB);
15       PUT EDIT(ZA,ZB,XA) (SKIP,A,A,A);
16       GOTO AA;
17 FIN : END;
```

Commentaires

La zone *ZL* sert à lire les données enregistrées sur cartes et la zone *XL* à lire le fichier indexé-séquentiel. La clé d'accès à ce fichier est le nom de la localité que l'on trouve dans *ZB* et que l'on transmet dans *XB* avant d'effectuer la lecture.

Prenons un exemple.
1) On a perforé sur une carte les informations suivantes :

DUPONT, JEAN LIEGE RUE DE L'UNIVERSITE

Lorsque l'ordinateur aura lu cette donnée dans la zone *ZL*, il devra transmettre le nom de la localité (LIEGE) dans la zone *XB*; ensuite, il faudra lire le fichier indexé-séquentiel

IND pour y trouver le numéro postal de LIEGE.

2) La zone *XB* contient :

LIEGEbbbbbbbbbbbbbbbbbbbbbbbbbbbb

C'est le nom de la localité suivi de 30 blancs (*XB* a 35 positions). La clé de lecture du fichier *IND* qui se trouve dans *XB* va conduire à l'enregistrement :

4000LIEGE

Le numéro postal se trouvera, après la lecture de *IND*, dans la zone *XA* qu'il suffira dès lors d'imprimer.

C. Mise à jour d'un fichier.

Un fichier indexé-séquentiel est généralement une liste de référence à laquelle on doit se reporter pour effectuer une tâche déterminée. Cette liste a normalement une certaine stabilité en ce sens que les informations qu'elle contient ne se modifient pas constamment. Toutefois, il peut arriver que certaines données doivent être changées sans que l'on touche cependant à la plupart d'entre elles. On peut alors utiliser un fichier de mise à jour (*UPDATE*) qui permet de lire les informations, de les modifier puis de les réécrire sur le disque magnétique.

La déclaration d'un fichier de mise à jour doit comprendre l'attribut *UPDATE* qui remplace *INPUT* et *OUTPUT.* Un fichier de mise à jour est donc un type de fichier intermédiaire entre les fichiers d'entrée et les fichiers de sortie.

DCL UP FILE RECORD UPDATE;

Le fichier *UP* est destiné à effectuer une mise à jour.

Un fichier *UPDATE* peut être utilisé, soit pour un fichier séquentiel, soit pour un fichier indexé-séquentiel; la seule obligation en matière de mise à jour est que le fichier à modifier *se trouve sur disques magnétiques.*

L'instruction de mise à jour est : *REWRITE FILE*(nom du fichier) *FROM*(nom de la zone);

Une instruction de réécriture *(REWRITE)* doit obligatoirement porter, dans la logique du problème, sur un enregistrement qui vient d'être lu sur disques et modifié dans

l'unité centrale de traitement.

Exemple

Nous prendrons, pour illustrer la mise à jour d'un fichier, le second exemple de fichier indexé-séquentiel que nous avons donné précédemment, c'est-à-dire le dictionnaire enregistré sur disques où les mots servent de clé et sont accompagnés d'une brève définition.

La tâche du programme que nous devons écrire consiste à modifier certaines définitions.

Pour effectuer ce travail, l'ordinateur utilisera deux fichiers :

1) le fichier indexé-séquentiel que l'on doit lire et modifier (nous l'appelerons *IND*);
2) un fichier séquentiel enregistré sur cartes qui contient les modifications à apporter; ce fichier devra contenir en plus des modifications, le mot proprement dit qui sert de clé à la lecture du fichier indexé-séquentiel (ce fichier se nommera *IN*).

Dans le fichier *IND*, on aura les informations suivantes :

la définition du mot dans les positions 1 à 50 et le mot dans les positions 51 à 80.
Les cartes du fichier *IN* auront une présentation rigoureusement identique.
On a enregistré dans *IND* la définition du mot *âge*; on trouve pour ce mot :

durée de la vie âge

On veut ajouter à la définition du mot *âge*, la notion "époque" de sorte que l'enregistrement modifié sera :

durée de la vie, époque âge

On doit trouver dans le fichier *IN* une carte qui porte ces indications et dont le contenu sera réécrit sur le fichier *IND* à l'endroit où est la définition du mot *âge*.

Le programme

```
1  EX : PROC OPTIONS(MAIN);
2  DCL 1 ZL,
3          2 ZA CHAR(50).
4          2 ZB CHAR(30);
5  DCL XL CHAR(80);
6  DCL IN FILE RECORD INPUT;
7  DCL IND FILE RECORD UPDATE KEYED ENV(INDEXED);
8  ON ENDFILE(IN) GOTO FIN;
```

```
 9 AA :READ FILE(IN) INTO(ZL);
10       READ FILE(IND) INTO(XL) KEY(ZB);
11       REWRITE FILE(IND) FROM(ZL);
12       GOTO AA;
13  FIN : END;
```

L'instruction 9 sert à lire une carte; cette carte porte une définition et le mot (la clé) que l'on doit modifier. L'opération 10 permet de lire l'enregistrement que l'on doit modifier dans le fichier indexé-séquentiel et enfin, l'ordre 11 assure la mise à jour de l'enregistrement sur le disque.

La gestion des fichiers que nous avons abordée dans ce chapitre est une tâche essentielle dans le traitement automatique des informations. C'est elle qui assure la bonne tenue des fichiers et qui permet de déterminer les procédés les plus efficaces d'accès aux informations. C'est, dans certains cas, une tâche difficile dont nous n'avons examiné que les aspects principaux. Enfin, c'est souvent une tâche fastidieuse puisqu'elle nécessite presque toujours l'accomplissement d'opérations routinières qui n'ont rien de commun avec les recherches des sciences humaines.

Je me suis limité volontairement aux notions principales de la gestion des fichiers de même que je m'étais limité aux instructions les plus importantes du langage PL/1. Il resterait beaucoup à dire si l'on voulait décrire de façon complète le traitement automatique des informations. Mais mon but n'était pas aussi ambitieux puisqu'il était de mettre à la portée des spécialistes des sciences humaines une méthode qui depuis vingt ans s'est étendue à tous les domaines du savoir et qui, d'ici quelques années, pourrait bien faire partie de la culture générale.

Au terme de ce volume, je voudrais encore faire deux remarques.

1) Un programme PL/1 pour être interprété et exécuté par l'ordinateur doit être accompagné de quelques cartes (les cartes-systèmes) qui ont pour rôle de faire traduire le programme PL/1 dans un langage compréhensible par la machine et qui permettent d'établir un lien entre les fichiers qu'un programme utilise et les supports où se trouvent ces fichiers. Si je n'ai pas donné la description de ces cartes, c'est parce que, en général, elles sont propres à un centre de calcul. Dès lors, chaque

utilisateur doit s'adresser à un informaticien pour obtenir la description des cartes-systèmes de l'ordinateur qu'il doit utiliser.

2) La lecture de cet ouvrage ne peut être qu'un début; elle doit fatalement conduire le lecteur à faire des exercices et même des recherches par ordinateur. Il ne servirait à rien de mémoriser la théorie du traitement de l'Information si cela ne débouchait sur un travail concret. Au reste, certaines instructions, certaines notions que nous avons exposées, peuvent encore paraître obscures, seules l'expérience et la pratique de la programmation pourront les éclairer.

NOTES BIBLIOGRAPHIQUES

ARSAC, Jacques, La Sciences informatique, Paris, Dunod, 1970.

LAURET, Annette, Principes de programmation des ordinateurs, 3e édition, Paris, Masson, 1972.

POULAIN, P., Eléments fondamentaux de l'informatique, 3e édition, Paris, Dunod, 1959.

OUVRAGES SUR LE PL/1

BERTHET, Charles, Le langage de programmation PL/1, Paris, Dunod, 1971.

I.B.M., PL/1 Checkout and Optimizing Compilers : Language Reference Manual.

VEILLON, Françoise; CAGNAT, Jean-Michel, Cours de programmation en langage PL/1, Paris, Colin 1971.

WEINBERG, G.M., Eléments de programmation en PL/1. Initiation et pratique. Traduit par Jean BERNARD, Paris, Dunod 1971.

TABLE DES MATIERES