FORMATION EN EDUCATION

RECHERCHE

PUBLICATIONS DU EFIS

Introduction aux bases de données relationnelles et à dBASE III.

2. dBASE III en mode conversationnel

André DELACHARLERIE

VERSION PROVISOIRE

PREMIERE PARTIE

DEPARTEMENT
EDUCATION & TECHNOLOGIE

Facultés Universitaires Notre-Dame de la Paix B-5000 Namur

AVANT-PROPOS

Nous avons découvert dans le fascicule 1 le monde des bases de données (BD) et des logiciels permettant de les gérer, appeler systèmes de gestion de bases de données (SGBD). Au travers de nombreux exemples, nous avons pu découvrir différentes façons d'organiser les informations stockées dans une BD. Parmi elles, la technique des bases de données relationnelles a retenu plus particulièrement notre attention et nous avons pu découvrir dans les grandes lignes les opérations qu'il convenait de mettre en oeuvre pour ajouter, modifier ou rechercher des informations.

Avec ce second document, nous allons concentrer notre regard sur le logiciel qui est actuellement le plus connu et le plus réputé pour la gestion des bases de données sur micro-ordinateur de type PC compatible. Que l'on ne s'y trompe cependant pas dBASE III, malgré sa très large diffusion n'est pas (ou plus) le logiciel le plus puissant et le plus performant du marché actuel. Il constitue néanmoins un très bon outil pour commencer à "faire" des bases de données.

Un troisième fascicule viendra, nous l'espérons, compléter cette série et s'intéressera plus particulièrement à l'emploi de dBASE en mode programmé ainsi qu'à la mise en oeuvre de BD complexes faisant intervenir un nombre important de fichiers.

Comme dans le premier document, notre but n'est pas d'être exhaustif mais bien d'offrir par le biais de nombreux exemples simples et compréhensibles de tous un "chemin" de découverte de ce logiciel. C'est donc intensionnellement que nous tairons certaines possibilités peu utilisées ou trop compliquées de dBASE. Si vous désirez plus de détails, vous pouvez toujours consulter l'un des nombreux ouvrages consacrés à dBASE et dont nous reprenons une sélection dans la bibliographie.

I. INTRODUCTION

1. Le monde dBase

Lorsque le logiciel dBase II, rédigé par Wagne Ratliff, commence à être distribué en 1980, par la société Ashton-Tate, il constitue une petite révolution dans le monde des micro-ordinateurs 8 bits. Il est, alors, le seul à offrir des outils de gestion de fichiers puissants et simples à utiliser qui peuvent être mis en oeuvre à la fois en mode interactif (l'utilisateur donne ses ordres au coup à coup et reçoit immédiatement les réponses de l'ordinateur) qu'en mode programmé (dBase est aussi un véritable langage de programmation comme BASIC ou PASCAL).

L'avènement des micro-ordinateurs IBM PC et compatibles donnera d'abord lieu à une réécriture de dBASE II pour ces nouvelles machines. Bientôt cependant, une nouvelle version du logiciel sera écrite afin d'exploiter au mieux la puissance des PC : ce sera, en 1984, dBASE III. Le langage de commande est conservé et enrichi et les performances sont largement améliorées (entre autres : 10 fichiers de données simultanés au lieu de deux, 1 milliard d'enregistrements par fichier au lieu de 65 000, ...).

Fin 1985, Asthon-Tate annonce une version nouvelle de son programme vedette. dBASE III Plus se distingue principalement de son prédécesseur par deux caractéristiques nouvelles. Premièrement, il est dorénavant capable de s'intégrer dans un réseau et de permettre ainsi le partage des données par plusieurs utilisateurs simultanés. Ensuite, son "interface utilisateur" a été entièrement revu. En effet, il est maintenant possible de réaliser un nombre important d'actions sur la base de données sans vraiment connaître la syntaxe des commandes dBASE. Des systèmes de menus, en français évidemment, permettent à l'utilisateur de préciser progressivement ses ordres jusqu'à ce que l'ordinateur les exécute.

Ces deux versions connaissent un grand succès. La concurrence ne reste pas inactive et produit une série de logiciels dont les caractéristiques sont fort semblables et qui se réclament généralement d'une grande compatibilité avec les fichiers de dBASE. Ce sont, entre autres, FoxBase + d'ABSoft, VP-Info de Paperback Software, dBXL de Wordteck Systems ou encore GEMJT Base 40.

Dans le cadre de ces notes, nous ferons en principe référence à dBase III en soulignant, le cas échéant, les différences ou les particularités propres à dBase III Plus. Ce choix est dicté par la raison bien simple, qu'au niveau des commandes principales, qui font l'objet de cette initiation, il n'existe pratiquement pas de différences entre les deux versions. Aussi, lorsque dans la suite, nous parlerons de dBASE, cela signifie que notre propos concerne aussi bien dBASE III que dBASE III Plus.

2. Mise en route de dBASE

A l'achat de dBASE, vous recevez plusieurs disquettes avec la documentation du logiciel portant, entre autres, les indications SYSTEM DISK # 1 et SYSTEM DISK # 2. La première de ces disquettes est protégée contre la copie et c'est pour cette raison que vous trouverez également un second exemplaire de cette disquette avec la mention SYSTEM DISK # 1 (BACK UP). En cas de destruction de la première disquette, cette copie de sécurité vous permettrait de continuer à utiliser dBASE. La disquette SYSTEM DISK # 2 n'étant pas protégée, il est vivement conseillé d'en réaliser immédiatement une copie de travail de façon à pouvoir ranger la disquette originale en lieu sûr.

Pour réaliser cette copie, introduisez une disquette contenant le système d'exploitation MS-DOS dans la première unité de lecture de disquette (en jargon micro-informatique, on parle du premier drive) et mettez l'ordinateur sous tension. Lorsque le symbole A> apparaît, frappez au clavier la commande (suivie de la

touche ↵ parfois marquée RETURN ou ENTER)

```
A>  DISKCOPY A: A:                                    (*)
```

Puis lorsque le micro-ordinateur vous le demandera, remplacez la disquette MS-DOS par la disquette SYSTEM DISK # 2 et pressez une touche du clavier. L'ordinateur lit alors le contenu de la disquette et le place dans sa mémoire centrale. Après quelques instants, l'ordinateur vous demande de placer la disquette copie. Remplacez donc la disquette 2 par une disquette vierge et pressez aussi une touche pour lancer l'écriture de cette dernière. Si la mémoire centrale de votre machine est inférieure à 640 K, il est possible que le programme demande d'échanger plusieurs fois les disquettes source (# 2) et copie (votre nouvelle disquette) pour conduire à bien l'opération de duplication. Pour terminer, il faut encore répondre N à la question qui vous demande si vous désirez faire une autre copie.

Tout est, à présent, prêt pour commencer à travailler. La séquence d'opération à réaliser pour lancer dBASE est la suivante.

* Démarrer (si ce n'est déjà fait) l'ordinateur avec une disquette MS-DOS.

* Dès l'apparition du symbole A>, placez la disquette # 1 dans le même drive et frappez au clavier :

  ```
  A>  DBASE
  ```

* Lorsque la phrase "Introduisez la disquette contenant le fichier dBASE.OVL ..." apparaît, remplacez la disquette # 1 par la copie de travail de la disquette # 2 que vous venez de réaliser, puis pressez une touche.

(*) Ce que vous devez taper sera dorénavant écrit en gras.

* dBASE est alors prêt, il affiche un texte rappelant l'interdiction de copie illicite du logiciel. La dernière ligne du texte est la plus courte (un point) mais est la plus importante : ce point nous indique que dBASE attend que nous lui donnions nos ordres. Nous retrouverons donc ce point chaque fois que dBASE aura exécuté une commande et attendra la suivante.

* Placez, enfin, votre disquette de données dans le second drive. Au début, bien entendu, il s'agira d'une disquette vierge qui aura seulement été formatée (*) mais bientôt celle-ci contiendra les différents fichiers de données que nous créerons ainsi que d'autres fichiers annexes pour la gestion de nos données (index, format de rapport, ...).
  Signalons, dès à présent, la commande qui permet de terminer une séance de travail dBASE. Il suffit pour cela de taper :

```
. QUIT
```

  pour retourner au niveau des commandes MS-DOS et de retrouver ainsi l'indicateur :

```
A>
```

3. Syntaxe générale d'une commande dBASE

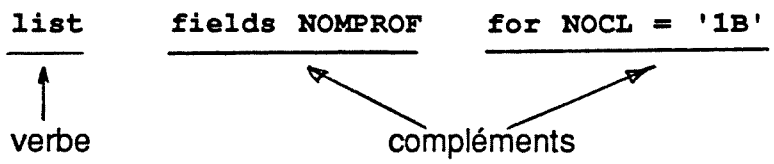
Bien que les différentes versions de dBASE soient disponibles en français le langage de commande continue à utiliser des mots d'origine anglaise. La traduction concerne la documentation qu'elle soit externe (le manuel) ou interne (message d'aide et d'erreurs).

(*) Pour formater une disquette vierge, il suffit de procéder comme pour la copie décrite plus haut mais en remplaçant la commande DISKCOPY A: A: par

```
A> FORMAT A:
```

et en veillant à bien placer une disquette neuve avant de presser une touche pour faire démarrer le formatage proprement dit).

Toute commande dBASE commence par un verbe et comporte éventuellement un ou plusieurs compléments qui précisent dans quelle limite l'action introduite par le verbe doit être réalisée. Revenons sur un exemple présenté page 29 dans le fascicule 1 :



Le verbe donne l'action à réaliser (ici lister le contenu d'une base de données) et les compléments précisent l'action en indiquant ici que, d'une part, seul le contenu du champ NOMPROF doit être affiché, et que d'autre part, il ne faut considérer que les enregistrements dont le champ NOCL est égal à 1B.

La syntaxe de dBASE veut que le verbe soit toujours placé en premier lieu alors que les compléments peuvent être donnés dans n'importe quel ordre. Nous verrons en temps utile quels sont les compléments qui sont acceptés par les différents verbes de dBASE. Notons encore que les commandes peuvent employer indifféremment les majuscules et les minuscules.

Mais arrêtons ici les considérations théoriques. Il est temps de commencer à donner des commandes à dBASE. Nous allons d'abord indiquer au logiciel que notre disquette de données se trouve dans le drive 2, appellé B: dans la terminologie MS-DOS.
La commande à taper après le point est :

```
. set default to B:
```

Cette action ne produit pas d'effet visible mais on peut s'assurer qu'elle a bien été exécutée en demandant quel est le contenu de la disquette. Pour cela, dBASE emploie le même ordre que le système d'exploitation MS-DOS :

```
. dir  *.*
```

et l'on obtient :

```
Aucun
     362496 octets disponibles sur l'unité
.
```

Cela est bien normal puisque la disquette placée en B: vient d'être formatée et ne compte donc aucun fichier. Il n'en est pas de même de la disquette dBASE qui se trouve dans le drive A:. Essayons ainsi :

```
. dir a:*.*
```

DIR et SET sont deux des nombreux verbes que dBASE connaît. Nous étudierons plus loin les différents compléments que l'on peut adjoindre (spécialement pour SET) mais voyons à présent deux verbes destinés à simplifier la vie de l'utilisateur dBASE.

4. A l'aide !

Le vocabulaire de dBASE, surtout de la version III Plus, est très riche, peut-être trop riche. Aussi il arrivera souvent à l'utilisateur d'hésiter sur la syntaxe exacte d'une commande. Heureusement dBASE est capable de l'aider grâce à la commande :

```
. help                                    (*)
```

(*) Pour les paresseux, il est aussi possible de frapper la touche marquée F1; on obtient le même résultat.

On obtient alors le menu principal du système d'aide et il suffit de sélectionner, grâce aux flèches ↓ et ↑ et à la touche ↵, le sujet sur lequel on désire être informé.

Cette méthode globale est cependant fastidieuse lorsque l'on cherche une information bien précise. Dans ce cas, on ajoutera en complément au "verbe" help le nom de la commande en question.

Par exemple :

```
. help list
```

donne l'écran suivant :

```
                                LIST
                                ____

Syntaxe      : LIST [<étendue>] [FIELDS <liste champs>] [FOR/WHILE <condition>]
               [OFF] [TO PRINT]
Description : Liste, sans pause périodique, le contenu d'un fichier
              (.dbf). Tous les enregistrements sont affichés si aucune
              spécification d'<étendue> n'est mentionnée.
              L'option FIELDS sélectionne les champs à lister.
              L'option OFF permet de ne pas afficher les  numéros
              d'enregistrement.

PgUp=écran préc., Esc=fin de HELP, ^Home=menu préc., ou frappez une commande.

                    CHOIX >
```

Arrêtons-nous sur quelques conventions d'écriture souvent employées dans la documentation du logiciel.

Dans la syntaxe d'une commande, les expressions écrites entre crochets **[ ]** sont des compléments facultatifs. Par exemple, le complément [TO PRINT] permet

d'envoyer les informations vers l'imprimante. S'il est omis, les informations seront écrites, par défaut, à l'écran. Les mots entre les signes < > désignent une partie obligatoire du complément mais qui est variable suivant le contexte. Enfin, / entre deux mots indique que l'on peut choisir un ou l'autre des deux mots en fonction bien entendu de l'objectif que l'on poursuit.

Par ailleurs, nous conserverons les mêmes conventions que dBASE pour désigner les touches remarquables et combinaisons de touches du clavier. Certaines de celles-ci ont une signification qui reste (à peu près) constante dans toutes les situations.

La touche **ESC** est certainement la plus employée, elle permet d'abandonner une activité en cours et de revenir à un niveau supérieur, c'est-à-dire le plus souvent de retrouver la situation où l'on se trouvait avant de lancer l'activité que l'on vient de frapper. C'est donc souvent la "sortie de secours", lorsque l'on vient de lancer une tâche par erreur.

A l'inverse, la combinaison des touches **CTRL** et **END**, pressées simultanément, (et notée par dBASE **^End**) permet de sortir d'une tâche mais, cette fois, en validant le travail effectué.
Nous verrons plus loin d'autres combinaisons faisant intervenir la touche CTRL; elles seront aussi notées avec le signe **^**. Par exemple, **^U** signifie, pressez simultanément la touche CTRL et la touche U.

dBASE III et surtout dBASE III Plus offrent un second moyen d'aide pour l'utilisateur novice : il s'agit du travail en mode "assisté". On entre dans ce mode en donnant la commande :

```
. assist
```

Il suffit, dès lors, de choisir dans les menus les actions que l'on désire réaliser. Dans la version Plus du logiciel, les possibilités de ASSIST sont sensiblement augmentées : ce mode de travail devient ainsi le plus commode pour l'utilisateur occasionnel de dBASE.

Nous ne nous étendrons cependant pas sur cette façon de manipuler le logiciel car notre objectif est avant tout de découvrir le langage de commande de dBASE et de nous préparer ainsi à programmer le logiciel pour que son emploi devienne totalement transparent à l'utilisateur, qu'il connaisse le logiciel ou non. Certains ouvrages cités dans la bibliographie décrivent en détail les possibilités offertes par le mode ASSIST.

5. Quelques Plus de dBASE III Plus

Nous en terminerons avec cette introduction en signalant encore deux caractéristiques propres à la version Plus et qui contribue à en rendre l'emploi plus agréable.

Il s'agit d'abord de la ligne de statut qui, en bas de l'écran et en mode inverse (lettres noires sur fond blanc) nous rappelle toute une série d'informations sur la situation actuelle. Cette ligne restera en permanence sur votre écran si vous tapez :

```
. set status on
```

De gauche à droite, on y trouve le nom de la commande, l'identification du disque, le nom du fichier et le numéro de l'enregistrement en cours de traitement. A droite, quatre indicateurs peuvent être affichés pour indiquer l'état de certaines touches. Par exemple, l'indication Min spécifie que l'on travaille en minuscule et que le blocage de majuscule (Caps Lock) n'est pas enclenché. Cette ligne de statut sera bien pratique, surtout en mode multifichier pour savoir dans quelle table on se

trouve et à quelle position exactement.

Un autre atout de dBASE III Plus , c'est son mode HISTORY. Il garde en mémoire les 20 dernières commandes frappées au clavier. Cette faculté est fort appréciable lorsque l'on commet une erreur de syntaxe dans l'énoncé d'une commande. Il suffit, de frapper sur la touche ↑ pour voir réapparaître la commande précédente. Il ne reste plus alors qu'à se déplacer dans la ligne avec → et ← pour aller corriger l'erreur. On peut aussi demander à l'ordinateur de lister les commandes précédemment tapées.
Essayer en donnant :

```
. list history
```

Nous verrons plus tard d'autres facilités offertes par ce logiciel mais étudions, à présent, le processus de création d'une base de données.

II. CREATION D'UN FICHIER

1. Enoncé du problème

Supposons que l'on doive créer une base de données reprenant la description des élèves inscrits dans une école.

L'analyse conceptuelle nous permet d'isoler les caractéristiques que nous retiendrons, à savoir :

Nom de l'élève : chaîne de 20 caractères
Prénom usuel : chaîne de 15 caractères
Nom et prénom du responsable légal : chaîne de 30 caractères
Adresse du domicile : décomposable en
- Rue et n° : chaîne de 30 caractères
- Code postal belge : chaîne de 4 caractères
- Nom de la localité : chaîne de 20 caractères
- N° de téléphone (y compris préfixe) : chaîne de 10 caractères

Date de naissance : date valide
Sexe : 1 caractère M ou F
Nombre de frères et soeurs : nombre entier positif ou nul
Seconde langue : 1 caractère avec le code
- N = Néerlandais
- A = Anglais
- L = Allemand

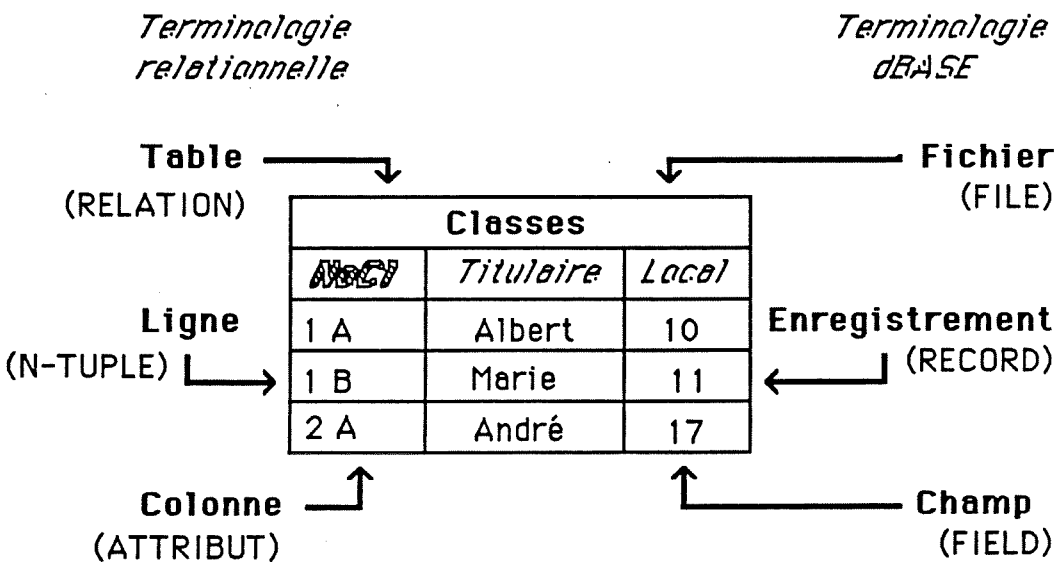
Intitulé de la classe : chaîne de 6 caractères (ex. : 3ECOA)
Doubleur : indicateur du type Oui ou Non
Commentaire : texte facultatif complétant l'information sur :
- les antécédents scolaires,
- la situation familiale,
- les problèmes de santé, ...

Pour ne pas compliquer notre approche, nous rassemblerons dans un premier temps toutes ces informations en une seule table. Nous verrons, plus tard, comment scinder celle-ci en deux pour isoler les nom et adresse du responsable légal dans une seconde table afin de normaliser notre base de données.

Voyons comment définir notre base de données et pour cela, commençons par nous habituer à la terminologie employée par dBASE.

2. Terminologie dBASE

Nous avons vu dans la fascicule 1 qu'une base de données relationnelle est composée d'une ou, bien souvent, de plusieurs tables. En dBASE, chaque table est sauvegardée sur disque en un fichier séparé. Aussi, dans la terminologie dBASE, le mot "fichier" (FILE) est pratiquement synonyme du mot "table". De même, au lieu des termes "n-tuple" ou "ligne", c'est le mot "enregistrement" (RECORD) qui a été retenu alors que "attribut" ou "colonne" est remplacé par "champ" (FIELDS).



3. Types de champ

Un coup d'oeil sur l'énoncé nous permet de constater qu'une bonne partie des champs seront des informations de type chaîne de caractères. Par contre, "Date de naissance" et "Nombre de frères et soeurs", bien que s'écrivant aussi avec des caractères, sont d'une autre nature et l'on est en droit d'espérer faire certaines "opérations" avec ces informations. Entre autre, on doit pouvoir comparer deux dates, par exemple pour lister les élèves nés après 01/01/72, ou calculer la moyenne du nombre de frères et soeurs. Pour cela, dBASE a prévu des types de champs adaptés à ces divers usages. En voici la liste :

*** Champs de type caractère**

C'est le type le plus courant. Un champ de type caractère peut recevoir toute combinaison des caractères du clavier : lettres majuscules et minuscules, chiffres, signes et espaces blancs. On doit toujours spécifier à la création du fichier la longueur maximum du champ. Pour le nom, on réservera, par exemple, de la place pour 20 caractères. Il est à noter qu'il s'agit là d'un maximum. Un nom de 17 caractères peut bien entendu y être placé; dBASE y ajoutera automatiquement 3 blancs (espaces) pour que le champ soit correctement rempli. Le nombre maximum de caractères pour un champ caractère est de 254.

*** Champs de type numérique**

Ce type de champ n'accepte que des nombres qui peuvent être entiers ou décimaux et qui peuvent compter jusque 19 chiffres (ou signes). On notera seulement que le signe - de négation et le point décimale nécessitent aussi chacun un caractère. Pour pouvoir écrire - 123.72, il faudra donc réserver au moins 7 chiffres dont 2 décimaux.
Nous verrons que ces champs pourront être utilisés comme données pour de nombreuses opérations mathématiques. Ils seront donc d'un grand secours pour

des applications de type comptable.

*** Champs de type date**

Ce type de champ est préformaté pour recevoir une date sous la forme 23/11/87. L'entrée d'information dans un tel champ n'est acceptée que si la date introduite est valide : cela nous évitera donc bien des fautes de frappe. Par ailleurs, une date est considérée par dBASE comme une sorte de nombre. Des opérations (nombre de jours entre deux dates par exemple) ou des comparaisons sont donc réalisées sans problème. Un champ date compte pour 8 caractères.

*** Champs de type logique**

Un champ de type logique est conçu pour les informations de type Vrai ou Faux. C'est, par exemple, le cas de notre champ "Doubleur ..." Ici, aussi, dBASE vérifie automatiquement que le caractère introduit est l'un des suivants. O pour "Oui", N pour "Non" mais aussi T pour "True" et F pour "False". La place occupée est de 1 caractère.

*** Champs de type mémo**

Le type mémo est fort différent des quatres types précédents. Il a été conçu pour recevoir des textes de longueur variable associés à certains enregistrements.
C'est le cas de notre information "Commentaire" : pour certains élèves, le commentaire peut être assez long (antécédents scolaires mouvementés du fait d'une situation familiale et d'une santé problématiques) alors que pour d'autres, il n'y a rien à signaler de spécial.
Réserver un ou plusieurs champs de type caractère n'est pas, dans ce cas, une bonne solution car tantôt la place réservée sera trop petite, tantôt elle sera complètement inutilisée.
Pour résoudre ce problème, Ashton-Tate a conçu un système, hélas un peu

complexe, qui fait intervenir un second fichier. Le texte, lorsqu'il y en a un, est stocké dans ce deuxième fichier tandis que l'on ne garde dans le fichier principal qu'une information de 10 caractères (en terme technique, on parle d'un pointeur) qui permet de retrouver le bon texte dans le fichier annexe.
Nous verrons cependant que ce type de champ est beaucoup moins simple d'emploi que les autres, il ne faudra donc pas en abuser.

Nous pouvons, à présent, attribuer à chaque champ le type qui lui convient. Par ailleurs, il faut aussi savoir que dBASE n'accepte comme nom de champ que des mots de maximum 10 caractères et ne comportant que des lettres ou des chiffres, et éventuellement le signe _ (soulignement) pour servir d'espace dans des noms composés. Le premier caractère doit être une lettre.

4. Création du fichier

Pour demander à dBASE de créer un nouveau fichier, on tape la commande CREATE suivie en complément du nom que portera ce fichier (max. 8 caractères). Ici, nous taperons donc :

```
. create ELEVES
```

On obtient un écran reprenant dans sa partie supérieure un aide mémoire décrivant les actions possibles (nous y reviendrons plus loin) et dans sa partie inférieure, l'ébauche du descriptif de notre fichier.
Il suffit de taper les noms des champs, leur type (les touches C, N, D, L ou M suffisent) et leur longueur en pressant la touche ↵ (RETURN ou ENTER) pour passer d'une case à l'autre.
On termine en frappant ^End (rappelons que cette notation signifie que l'on doit presser simultanément les touches CTRL et End).

L'écran obtenu est par exemple :

	Nom champ	Type	Dim	Dec		Nom champ	Type	Dim	Dec
1	NOM	Caractère	20		9	SEXE	Caractère	1	
2	PRENOM	Caractère	15		10	FRERES	Numérique	2	0
3	RESP_LEGAL	Caractère	30		11	SEC_LANG	Caractère	1	
4	ADRESSE	Caractère	30		12	CLASSE	Caractère	6	
5	CODEPOSTAL	Caractère	4		13	DOUBLEUR	Logique	1	
6	LOCALITE	Caractère	20		14	COMMENTAIR	Memo	10	
7	TELEPHONE	Caractère	10						
8	NAISSANCE	Date	8						

Frappez RETURN pour confirmer. Une autre touche pour abandonner.

Lorsque l'on est revenu au symbole d'attente de dBASE (le point), il est intéressant de vérifier la structure du fichier. Pour cela, on frappe

```
. display structure
```

ou plus simplement encore, on presse la touche F5 et l'on obtient :

```
Structure du fichier     : B:eleves.dbf
Nombre d'enregistrements :        0
Dernière mise à jour     : 21/08/87
Champ    Nom champ    Type          Dim   Dec
    1    NOM          Caractère      20
    2    PRENOM       Caractère      15
    3    RESP_LEGAL   Caractère      30
    4    ADRESSE      Caractère      30
    5    CODEPOSTAL   Caractère       4
    6    LOCALITE     Caractère      20
    7    TELEPHONE    Caractère      10
    8    NAISSANCE    Date            8
    9    SEXE         Caractère       1
   10    FRERES       Numérique       2
   11    SEC_LANG     Caractère       1
   12    CLASSE       Caractère       6
   13    DOUBLEUR     Logique         1
   14    COMMENTAIR   Memo           10
** Total**                         159
```

Il est bien pratique de garder cette table à portée de la main. Aussi, pour la faire imprimer sur papier, nous relancerons la même commande avec le complément TO

PRINT.

```
. display structure to print
```

Attention, cependant, vérifiez bien que l'imprimante est correctement branchée et qu'elle est "on line" sinon dBASE risque de se bloquer car il ne supporte pas de "parler à une imprimante qui ne l'écoute pas".

Remarques

1. Si vous êtes curieux, vous aurez peut-être compté que la place occupée par les champs d'un enregistrement est de 158 caractères et non 159 comme on peut le lire avec la commande display structure. En fait, dBASE ajoute toujours un caractère qui servira comme nous le verrons plus loin, à gérer les enregistrements que l'on désire supprimer.

2. Un enregistrement ne peut comporter plus de 128 champs et son total de caractères ne peut dépasser 4 000. Cela est donc bien suffisant surtout lorsque l'on a correctement normalisé sa base de données. (*) Signalons aussi qu'un fichier dBASE peut compter jusque 1 milliard d'enregistrements. Inutile de dire que c'est plutôt la taille des disquettes où même du disque dur qui sera notre limite.

3. La taille d'un texte enregistré par l'intermédiaire d'un champ mémo occupe au minimum 512 caractères et peut normalement s'étendre jusqu'à 5 000 caractères (± 3 pages de texte).

(*) On a vu dans le fascicule 1 que la normalisation des relations aboutissait toujours à scinder les tables en tables de plus petites dimensions.

4. Le fichier principal sera sauvegardé sur la disquette avec le nom que vous aurez donné et il sera complété par l'extension .DBF tandis que le fichier annexe, contenant les champs mémos portera le même nom avec l'extension .DBT.
   On peut le vérifier en tapant :

```
. dir *.*
```

5. Réutilisation d'un fichier

Lors d'une session de travail en dBASE, nous pouvons être amenés à utiliser plusieurs fichiers différents qui auront été créés antérieurement. La manoeuvre qui signale au logiciel que l'on désire commencer à travailler avec un fichier se nomme : l'ouverture du fichier. Tout naturellement, la manoeuvre inverse s'appelle la fermeture du fichier. Cette dernière action est importante lorsque l'on a terminé un travail sur un fichier car elle permet d'être sûr que toutes les mises à jour demandées ont bien été écrites, aussi bien l'ouverture que la fermeture d'un fichier. Pour ouvrir le fichier ELEVES, on tapera :

```
.use ELEVES
```

tandis que pour fermer le fichier actif, on donnera simplement :

```
. use
```

lorsque nous aurons plusieurs fichiers de données ouverts simultanément (dBASE en accepte jusque 10), on pourra les fermer d'un seul coup en tapant :

```
. close databases
```

Vous pouvez essayer ces commandes et vérifier que le fichier est ouvert ou non en employant DISPLAY STRUCTURE.

6. Modification de la structure d'un fichier

Si vous avez commis certaines erreurs lors de la création du fichier (nom de champ mal orthographié, dimension incorrecte, champ oublié, ...); cela n'est pas bien grave car il existe une commande qui permet la correction de la structure du fichier. Elle ne peut cependant être utilisée que lorsque le fichier est ouvert. On tapera donc :

```
. use ELEVES                    (si ELEVES n'est pas déjà ouvert)
. modify structure
```

On retrouve alors le même écran que lors de la création du fichier. Il nous suffit de regarder la partie haute de l'écran pour voir les commandes que nous pouvons employer pour nous déplacer dans la structure et pour y apporter les modifications désirées.

Les quatres touches fléchées → ← ↑ ↓ permettent de déplacer le curseur sur n'importe quelle zone de la table de description. La combinaison ^N (pour rappel CTRL + N) permet d'insérer un nouveau champ dans la liste tandis que ^U sera employé pour supprimer un champ. Enfin les (traditionnels) ^End et Esc permettent respectivement de clôturer en sauvegardant les modifications où d'abandonner sans mémoriser les changements effectués.

Exercices

1. Modifier la structure précédente de façon à ce que :
   - le prénom ait une dimension de 20 caractères;
   - le code postal soit de type numérique;

- le champ FRERES devienne FRER-SOEUR;
- le champ CLASSE disparaisse et soit remplacé par deux champs;

SECTION	4 caractères
ANNEE	numérique 1 caractère

Imprimer alors la nouvelle structure sur papier.

2. Restituer la situation de départ.

Remarques

1. La modification de structure d'un fichier est aussi possible lorsque celui-ci comporte des enregistrements. Les informations des champs correspondant sont alors recopiées, au mieux, dans la nouvelle structure. Il n'y a cependant pas de recopie d'information si l'on change en même temps le nom et la dimension d'un champ. Dans ce cas, il convient d'opérer en deux étapes.

2. La version antérieure du fichier n'est pas perdue, elle subsiste sur la disquette avec le même nom et l'extension .BAK.

III. INTRODUCTION DES DONNEES

1. Ajout d'un enregistrement

Notre but en créant le fichier ELEVES est bien entendu d'y stocker des informations sur un certain nombre d'élèves. La commande dBASE qui nous permet d'ajouter de nouvelles fiches se nomme APPEND.

```
. append
```

dBASE crée alors un écran où l'on trouve, en haut le numéro de l'enregistrement que nous allons remplir. C'est donc ici le premier. Ensuite, un cadre nous indique quelles sont les combinaisons de touches que l'on peut employer à ce niveau et enfin la partie basse de l'écran nous donne la liste des champs que nous avons définis lors de la création de la structure.
Chaque champ est suivi d'une zone blanche (en mode inverse) dimensionnée à la longueur que nous avons prévue et qui est destinée à recevoir les informations. Le curseur clignote au début de la première zone.
Nous pouvons donc commencer à taper le nom, puis le prénom, ... Dès qu'une information est complètement écrite dans un champ, on passe au début du champ suivant avec la touche ↵. Les touches flèches droite et gauche nous permettent, le cas échéant, de corriger le contenu du champ dans lequel on se trouve tandis que les flèches haut et bas permettent de passer d'un champ à l'autre.
A propos des corrections, il faut savoir que dBASE, comme de nombreux autres logiciels, admet deux modes d'écriture : le mode insertion et le mode recouvrement.

Voyons cela avec un petit exemple. Tapons LECOMTE dans la zone NOM. Supposons qu'il fallait écrire LE COMTE en deux mots. Pour corriger cela, il nous faut revenir sur la lettre C en pressant quelques fois la touche ←. Lorsque le curseur clignote sous le C, on frappe la barre d'espace pour ajouter un blanc. Que

se passe-t-il ? La lettre C disparaît et est recouverte par l'espace. Il s'agit bien du mode recouvrement : tout caractère frappé recouvre celui qui se trouvait à cet endroit. C'était déjà le cas auparavant mais comme les caractères frappés recouvraient des blancs, on ne pouvait pas s'en rendre compte.

Nous voilà donc avec le nom LE OMTE. Pour achever la correction, une première solution consiste à retaper toute la fin du nom qui recouvrira l'ancienne version. dBASE nous offre une seconde solution, c'est d'insérer le caractère C manquant entre l'espace et la lettre O. Pour passer en mode insertion, il suffit de presser une fois la touche marquée Ins (sous O du pavé numérique) (*).
Maintenant, il ne nous reste plus qu'à frapper le caractère C pour que celui-ci vienne s'intercaler à l'endroit du curseur en repoussant vers la droite le reste du mot.

La présence du mot INSERTION dans la ligne supérieure de l'écran nous permet à tout moment de savoir dans quel mode on se trouve. (**)

Pour l'effacement de caractères erronés, on dispose aussi de deux moyens. Le premier, avec la touche ← (souvent au coin supérieur droit du clavier alphabétique) permet d'effacer le caractère qui se trouve à gauche du curseur. Tandis que la touche DEL supprime le caractère recouvert par le curseur. Dans les deux cas, à l'inverse de ce qui se passait avec l'insertion, le texte à droite du caractère supprimé se déplace pour venir recouvrir le caractère disparu.

(*) Si cela provoque l'apparition d'un zéro à l'écran, il suffit de presser d'abord sur la touche Num Lock (parfois Func Lock) avant de presser à nouveau sur Ins.

(**) En dBASE III Plus, si la ligne de statutest active, c'est dans la partie droite de cette ligne qu'apparaît, le cas échéant, le mot "Ins".

Pour vous faire une bonne idée de l'utilisation de ces touches, tapez un nom compliqué et assez long, puis modifiez-le en employant les différentes touches dont nous venons de parler.

Remarques

1. Comme prévu les champs de type numérique, date et logique sont déjà préformatés pour les informations à recevoir : les "/" (prononcez "slash") séparent les jours, mois et année d'une date, tandis que le point est déjà placé dans le cas d'un champ numérique comportant des décimales. De plus, les caractères indésirables (lettre dans les dates par exemple) ne sont pas acceptés et font retentir un petit bip sonore.

2. Dans les champs de type caractère, l'utilisateur peut employer aussi bien les majuscules que les minuscules. Nous verrons cependant que les manoeuvres de recherches seront facilitées si les champs sur lesquels elles portent ne comportent que des majuscules. Il faut, en effet, savoir que a priori dBASE considère que DUPOND, Dupond et dupond sont trois informations différentes. dBASE offre bien entendu des outils pour régler ces petits problèmes. Nous les aborderons plus tard. En attendant, utilisons en général des majuscules et réservons les minuscules pour les champs moins susceptibles de faire l'objet de critères de recherche (le nom de la rue, par exemple) ainsi que pour les textes du champ mémo.

3. L'écriture d'information dans le champ mémo se réalise de façon assez différente. On place le curseur sur le mot "mémo", puis on presse ^Home. (Home se trouve sur la touche 7 du pavé numérique). On obtient alors un écran presque vide : il s'agit du mini-traitement de texte de dBASE. On peut alors écrire là le texte désiré. Ici aussi, on peut employer toutes les touches que nous

avons décrites un peu plus haut. (*) Un tableau rappelant la liste de ces touches peut d'ailleurs être obtenu dans la version Plus en pressant F1. Lorsque tout le texte est entré, on le sauvegarde en pressant ^End.

Le contenu de l'enregistrement apparaît à présent comme ceci :

NOM	**LECOMTE**
PRENOM	**CECILE**
RESP_LEGAL	**LECOMTE MARCEL**
ADRESSE	**Rue de la Station, 73**
CODEPOSTAL	**5730**
LOCALITE	**MALONNE**
TELEPHONE	**081/363432**
NAISSANCE	**23/05/74**
SEXE	**M**
FRERES	**2**
SEC_LANG	**N**
CLASSE	**3TSE**
DOUBLEUR	**N**
COMMENTAIR	memo

Pour passer à l'enregistrement suivant, on se place sur le dernier champ (ici mémo) et l'on presse RETURN ou ↵. Une nouvelle fiche vierge apparaît alors avec le numéro d'enregistrement suivant.

Pour quitter l'ajout d'enregistrement, on presse ^End (encore lui) ou on laisse vide le premier enregistrement de la fiche vierge suivante (en pressant RETURN). On retrouve alors le point qui nous dit que dBASE attend une autre commande.
On peut redonner la commande APPEND autant de fois que l'on veut : les

(*) D'autres actions sont possibles; elles utilisent des combinaisons de touches rappelant celles du traitement de texte WordStar. Elles ne sont cependant pas indispensables au début aussi n'en parlerons-nous pas à ce moment-ci.

nouveaux enregistrements viennent toujours s'écrire à la suite des précédents (en faisant donc croître le numéro d'enregistrement).

Exercice

Remplissez ainsi quelques fiches, fermez le fichier, demandez à voir la liste des fichiers de données de la disquette avec la commande :

```
. dir
```

puis rentrez à nouveau dans le fichier pour réajouter une ou deux autres fiches. Vous trouverez dans l'annexe 1 les informations qui constituent le fichier ELEVES que nous emploierons par la suite.

2. Insertion d'enregistrement

Il est possible que nous désirions garder les enregistrements dans un certain ordre. On pourra donc être amené à vouloir ajouter une nouvelle fiche, non pas à la fin du fichier, mais à un endroit précis, entre deux fiches existantes. Cela est possible en dBASE avec la commande INSERT. Par exemple, pour ajouter une flèche entre les enregistrements 3 et 4, il suffit de se placer sur la fiche 3 avec la commande GOTO (= allez à ...), puis de demander l'insertion de la fiche nouvelle.

```
. goto 3
. insert
```

On obtient alors le même type d'écran qu'avec APPEND mais après avoir rempli la fiche, on revient directement au mode commande. Pour insérer plusieurs fiches, il faut donc donner autant de fois la commande INSERT.

Remarque importante

L'usage de la commande INSERT n'est pas recommandé car d'une part, on étudiera d'autres moyens, plus simples, pour ordonner les fiches de notre base de données et d'autre part, l'insertion d'une fiche oblige dBASE à recopier, sur le disque, toutes les fiches qui suivent.
Si le fichier est un peu important, cette opération peut donc demander plusieurs dizaines de secondes voire quelques minutes.

IV. AFFICHAGE DES DONNEES

1. Interrogation simple

Nous venons de placer une petite série d'enregistrements dans notre fichier. Il est tout naturel de vouloir contempler le résultat de notre travail en demandant à dBASE d'afficher le contenu du fichier ELEVE. Cela est très facile avec la commande :

```
. list
```

Le résultat obtenu n'est malheureusement pas à la hauteur de nos espoirs. En effet, faute d'indication de notre part, dBASE a affiché toutes les informations qu'il connaissait en les écrivant à la queue leu leu en passant à la ligne chaque fois qu'il rencontre le bord droit de l'écran.

En pratique, on ne s'intéressera qu'à une partie des champs et l'on réalisera donc une projection de la relation (*). Demandons ainsi la liste des nom, prénom et classe, on aura : (**)

```
. list fields NOM, PRENOM, CLASSE
```

Enreg.	N°	NOM	PRENOM	CLASSE
	1	LECOMTE	CECILE	3TSE
	2	LACROIX	MARIE	2CO
	3	GYSEL	MARC-ALEXANDRE	3TSL
	4	BRASSEUR	ALAIN	1A
	5	RIVIELLARO	SAMINA	3TSE
	6	DEFLEUR	ERIC	2CO
	7	DELVAUX	LAURA	3TSL
	8	JACQUART	ANNE-CHRISTINE	1A
	9	LUCAS	BENOIT	2CO

(*) Cfr fascicule 1, chapitre V.

(**) Tous les exemples donnés ici font référence à un fichier ELEVES de 9 enregistrements dont le contenu est détaillé dans l'annexe I.

On reconnaît ici le complément FIELDS qui introduit les noms des champs à afficher. Ces noms peuvent être donnés dans un ordre différent de celui où ils ont été introduits dans la structure du fichier. Le mot FIELDS est facultatif dans le cas de la commande LIST (et dans ce cas uniquement).

D'autre part, si l'on n'a pas besoin du numéro d'enregistrement on peut demander à dBASE de l'omettre en ajoutant le complément OFF comme dans l'exemple que voici :

```
. list off CLASSE, PRENOM, NOM, ADRESSE
```

CLASSE	PRENOM	NOM	ADRESSE
3TSE	CECILE	LECOMTE	Rue de la Station, 73
2CO	MARIE	LACROIX	Rue de la Libération, 2a
3TSL	MARC-ALEXANDRE	GYSEL	Place du Petit Pont, 3, bte 2
1A	ALAIN	BRASSEUR	Avenue de l'Abbaye, 45
3TSE	SAMINA	RIVIELLARO	Rue des Vignerons, 12
2CO	ERIC	DEFLEUR	Rue de la Rose, 55
3TSL	LAURA	DELVAUX	Chaussée des Pavés, 5
1A	ANNE-CHRISTINE	JACQUART	Cité des Rossignols, 30
2CO	BENOIT	LUCAS	Avenue Cardinal Mercier, 17

Continuons à étudier la commande LIST : nous allons voir qu'elle accepte un bon nombre de compléments différents. Cela peut paraître un peu ennuyeux et difficile à retenir mais il faut savoir que LIST est la principale commande d'interrogation de la base de données en mode direct.
Nous nous en servirons donc souvent.

2. Notions d' "étendue"

Par défaut, si nous ne précisons rien, dBASE applique la commande LIST à tous les enregistrements du fichier en cours. En fait, il agit comme si nous avions donné le complément ALL qui affecte "l'étendue" (certains auteurs parle de la

"portée" de la commande) à tout le fichier.

On peut bien entendu réduire cette portée en spécifiant le numéro de l'enregistrement que l'on désire examiner à la suite du complément RECORD :

```
. list record 3 NOM,PRENOM,TELEPHONE
Enreg.  N° NOM          PRENOM           TELEPHONE
       3   GYSEL        MARC-ALEXANDRE   081/223344
```

Si l'on veut plusieurs enregistrements, on emploiera le complément NEXT en le faisant suivre du nombre d'enregistrements souhaités.

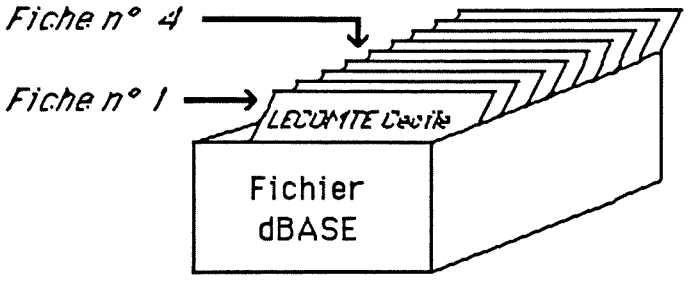
```
. list next 4 NOM, PRENOM, TELEPHONE
Enreg. N°     NOM          PRENOM           TELEPHONE
       3      GYSEL        MARC-ALEXANDRE   081/223344
       4      BRASSEUR     ALAIN            081/434413
       5      RIVIELLARO   SAMINA
       6      DEFLEUR      ERIC             081/734598
```

dBASE affiche donc les champs demandés pour les 4 enregistrements suivants en commençant par l'enregistrement sur lequel on se trouve.
Ceci nous amène à dire quelques mots de l'organisation du fichier dBASE.

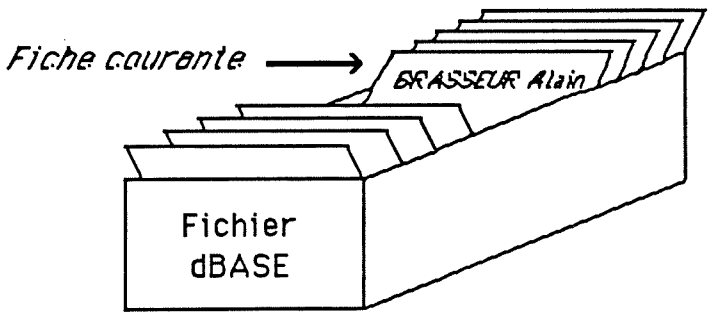
3. Organisation physique du fichier

On a pu constater que le logiciel attribue automatiquement un numéro d'enregistrement à chaque fiche et ceci au fur et à mesure des ajouts effectués avec APPEND. Ce numéro détermine la localisation physique de l'enregistrement sur le support magnétique (disquette ou disque dur). Tout se passe en fait comme avec des fiches cartonnées dans une boîte. La première fiche remplie est placée au début, puis les suivantes sont placées derrière. Chaque fiche possède donc un numéro d'ordre physique : la fiche numéro 4, c'est le quatrième carton que l'on

trouve dans la boîte en commençant à compter à partir du bord avant. Cette numérotation physique est nécessairement continue : si l'on enlève la quatrième fiche et qu'on la jette, il y reste une quatrième, c'est l'ancienne cinquième fiche (sauf, bien entendu, s'il n'y avait que quatre fiches).



Lorsque dBASE exécute une commande sur le fichier, il ne manipule jamais qu'une seule fiche à la fois. On l'a bien vu avec les opérations d'ajout (APPEND et INSERT). Cela reste vrai avec les autres commandes, même pour LIST ALL. En effet, dBASE pour afficher tout le fichier, commence par afficher les champs du premier enregistrement, PUIS il fait de même avec le second, PUIS avec le troisième PUIS ... jusqu'à ce qu'il ait passé toutes les fiches en revue (*).
A un instant donné, dBASE ne regarde donc qu'une seule fiche, nous l'appellerons la FICHE COURANTE, son numéro physique est à tout moment connu de dBASE et est stocké dans ce que l'on appelle le POINTEUR.



(*) Si vous avez déjà fait un peu de programmation vous reconnaîtrez là un processus répétitif.

On peut demander au logiciel de nous montrer le contenu de ce pointeur avec la commande (*) :

```
. ? recno()
          6
```

qui nous répond que la fiche courante porte le numéro 6 : c'est en effet la dernière que nous avions manipulé avec la commande LIST précédente. Notons que le "verbe" employé ici est **?** et qu'il signifie "Affiche". Le complément, "recno ()" indique ce que l'on doit afficher, à savoir, le NumérO de RECord.

Nous avons déjà dit que la commande GOTO permettait d'obliger dBASE à se déplacer dans le fichier et à changer de fiche courante comme le montre la séquence de commandes suivantes

```
. goto 2
. ? recno()
          2
. list next 1 NOM,PRENOM
Enreg. N°     NOM                    PRENOM
       2      LACROIX                MARIE
```

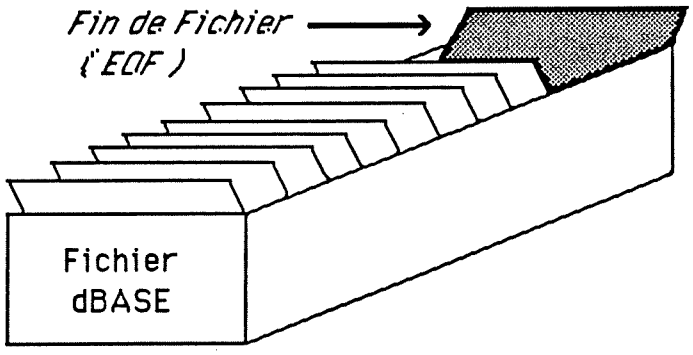
Voyons, à présent, ce qui se passe lorsqu'on arrive à la fin du fichier. Pour y arriver sûrement, exécutons une commande LIST sur tout le fichier, puis demandons à voir la valeur du pointeur.

(*) En dBASE III Plus, la valeur du pointeur est constamment indiquée au milieu de la ligne de status sous la forme "Enr : 6/9" qui indique que c'est la 6ème fiche d'un fichier qui en compte 9 au total.

```
.list all NOM
Enreg. N°    NOM
       1     LECOMTE
       2     LACROIX
       3     GYSEL
       4     BRASSEUR
       5     RIVIELLARO
       6     DEFLEUR
       7     DELVAUX
       8     JACQUART
       9     LUCAS
. ? recno()
         10
. list next 1 NOM
Enreg. N° NOM
.
```

Que se passe-t-il donc, notre fichier ne compte que 9 enregistrements, pourtant le pointeur est passé outre et s'est arrêté avec la valeur 10. Il n'y a pourtant pas vraiment de fiche 10 comme on peut le constater en demandant d'afficher le NOM de la fiche courante.

En reprenant notre comparaison avec la boîte de fiches, tout se passe comme si l'on avait, après les "vraies" fiches, un carton de couleur rouge ne comportant aucune inscription. Ce carton rouge nous indique ainsi, lorsqu'on le rencontre, que l'on se trouve à la fin du fichier.



Cette fiche est bien vide (ne contient pas d'information) et porte toujours le numéro qui suit celui de la dernière fiche valide. dBASE nous offre d'ailleurs une commande qui nous permet de poser la question :

" Vois - tu la fiche rouge ? "

C' est :

```
. ? eof(  )
.T.
```

EOF ( ) , abréviation de End Of File (= Fin de Fichier) désigne l'information que nous cherchons : ce sera soit .T. (pour True = Vrai) qui signale que l'on est bien à la fin du fichier ou .F. (pour False = Faux) qui indique au contraire que l'on n'y est pas. On peut d'ailleurs le vérifier en donnant :

```
. goto 9
. ? eof( )
.F.
```

Plus tard, lorsque nous apprendrons à programmer avec dBASE, nous utiliserons souvent les outils (appelés "fonctions") RECNO ( ), EOF ( ), et bien d'autres. D'ici là, ils nous serviront seulement de temps en temps à savoir où nous sommes dans la base de données.

4. Sélection d'enregistrements

Supposons, à présent, que nous désirions extraire de notre petit fichier les nom et prénom des élèves qui sont inscrits en 2CO. Pour cela, nous réaliserons une sélection sur notre table (*) en ajoutant un complément à notre commande LIST pour spécifier la CONDITION de sélection. Les élèves inscrits en 2CO sont ceux POUR lesquels le champ CLASSE contient la chaîne '2CO'.

La condition se traduit assez naturellement en dBASE par :

for CLASSE = '2CO'

(*) Revoir au besoin le fascicule 1, chapitre V.

et la commande complète devient :

```
. list NOM, PRENOM for CLASSE = '2CO'
Enreg.    N°  NOM              PRENOM
        2     LACROIX          MARIE
        6     DEFLEUR          ERIC
        9     LUCAS            BENOIT
.
```

On constate bien que le logiciel a passé toutes les fiches en revue en commençant par le début du fichier (la portée par défaut est ALL) et n'a affiché les champs demandés que pour les enregistrements vérifiant la condition posée.

dBASE est capable de reconnaître un grand nombre d'énoncé de condition pour peu qu'il respecte certaines règles syntaxiques.

Nous nous bornerons ici à donner les règles de base et à présenter les outils (fonctions) les plus importants. Nous en étudierons d'autres dans le chapitre VI consacré à la recherche de données.

5. Enoncé des conditions

Pensons à quelques conditions en langage naturel que nous pourrions poser à propos de notre fichier d'élèves. On pourrait ainsi chercher les élèves qui ...

a) portent le nom BRASSEUR;
b) n'habitent pas NAMUR ville;
c) font partie d'une famille nombreuse;
d) sont dans une classe de 3ème;
e) n'ont pas le téléphone;
f) sont des filles de 1A.

dBASE ne peut évidemment faire sa sélection que sur base des informations qu'il possède dans ses fichiers. Il nous faut donc chaque fois déterminer quel(s) champ(s) vont permettre à dBASE de décider si l'enregistrement vérifie ou non la condition.
Cela peut être évident comme pour la condition a) où c'est le champ NOM qui devra être examiné. Dans d'autres cas, il faut chercher un peu plus pour déterminer comment on peut déduire l'information cherchée. Pour la condition c), ce serait très simple si nous disposions d'un champ de type logique (OUI ou NON) qui s'appelerait, par exemple, FAM-NOMBR. Ce n'est pas le cas mais on peut facilement déduire cette information du nombre de frères et soeurs :

Les élèves qui font partie d'une famille nombreuse.

Nombre total d'enfants est supérieur ou égal à 3.

Nombre de frères et soeurs est supérieur ou égal à 2.

Nous avons aussi isolé le champ qui servira à la comparaison, ainsi que l'information qui fera référence. Il ne reste qu'à préciser l'opération de comparaison. Le plus simple et le plus classique est l'égalité représenté par =. La condition a) s'énoncera donc :

NOM = 'BRASSEUR'

et l'on pourra écrire par exemple :

```
. list NOM, PRENOM, TELEPHONE for NOM = 'BRASSEUR'
Enreg. N°    NOM                PRENOM          TELEPHONE
       4     BRASSEUR           ALAIN           081/434413
```

Le lecteur aura constaté que la chaîne de caractères qui sert de référence est située entre deux DELIMITEURS. Ici, c'est l'apostrophe (') qui sert de délimiteur, mais dBASE accepte aussi d'autres délimiteurs et en particulier les guillemets ("). Ces derniers seront indispensables pour délimiter sans équivoque la chaîne de caractères "Avenue de l'Abbaye".

Notons à ce propos que dBASE, comme bien d'autres logiciels, n'accepte de comparer que des "choses" comparables : le type de l'expression de référence doit être le même que celui du champ à examiner.
Ainsi, la condition

CODEPOSTAL = 5000

n'est pas valide et doit être remplacée par

CODEPOSTAL = '5000'

Par contre,

FRERES = 1

est une condition syntaxiquement correcte car FRERES est de type numérique comme le constante 1 (et non '1').

Pour exprimer en dBASE la condition b) "n'habite pas NAMUR-ville" nous avons le choix entre deux solutions. Soit employer l'opérateur <> qui signifie "est différent de" (*) et l'on obtient :

```
. list NOM, PRENOM, LOCALITE for LOCALITE <> 'NAMUR'
Enreg. N°   NOM              PRENOM           LOCALITE
      1     LECOMTE          CECILE           MALONNE
      2     LACROIX          MARIE            SAINT-SERVAIS
      4     BRASSEUR         ALAIN            FLOREFFE
      6     DEFLEUR          ERIC             SAINT-SERVAIS
      7     DELVAUX          LAURA            MALONNE
      8     JACQUART         ANNE-CHRISTINE   JAMBES
```

(*) dBASE accepte aussi le symbole # pour le même usage que <>.

Soit demander à dBASE de considérer le contraire (on dit la négation) de la condition LOCALITE = 'NAMUR' en utilisant l'opérateur .NOT.
On aurait ainsi :

```
. list NOM, PRENOM, LOCALITE for .not. LOCALITE = 'NAMUR'
```

qui produirait le même résultat.

L'opérateur utilisé pour la condition c) est le "supérieur ou égal " bien connu des mathématiciens. Il s'exprime en dBASE par >= et l'on a la commande :

```
. list NOM, PRENOM, FRERES for FRERES >= 2
```

Enreg. N°	NOM	PRENOM	FRERES
1	LECOMTE	CECILE	2
5	RIVIELLARO	SAMINA	4
7	DELVAUX	LAURA	2

Pour la condition d), il nous faut être attentif à ce que notre fichier renferme 2 classes de 3ème : la 3TSE (Sciences-Eco.) et 3TSL (Sciences-Labo). Ici aussi, on peut exprimer la condition de plusieurs manières.

Premièrement, on peut dire que le 1er caractère du nom de la classe doit être '3'. Pour cela, nous avons besoin d'un "outil" pour extraire le 1er caractère du champ CLASSE. Il s'agit de la fonction :

SUBSTR (CH,D,LG)

où CH est la chaîne dont on extrait une partie, D est le numéro du premier caractère à considérer et LG le nombre de caractère à sortir. Ici, nous désirons obtenir 1 caractère à partir du 1er de la chaîne contenue dans le champ CLASSE. La condition s'écrira donc

substr (CLASSE, 1, 1) = '3'

et la commande complète :

```
. list NOM, PRENOM, CLASSE for substr(CLASSE,1,1) = '3'
Enreg.  N°  NOM                PRENOM            CLASSE
        1   LECOMTE            CECILE            3TSE
        3   GYSEL              MARC-ALEXANDRE    3TSL
        5   RIVIELLARO         SAMINA            3TSE
        7   DELVAUX            LAURA             3TSL
```

Une seconde idée de solution, c'est de penser que le nom de la classe ne comporte de toute façon qu'un seul chiffre. Il suffit donc de voir si le caractère '3' est contenu (n'importe où) dans le champ CLASSE. dBASE dispose, en effet, de l'opérateur $ qui signifie "est contenu dans ". On peut ainsi écrire :

```
. list NOM, PRENOM, CLASSE for '3' $ CLASSE
```

L'opérateur $ ne fonctionne bien entendu que pour comparer des chaînes de caractères.
Notons cependant qu'avec cette condition, la classe 1A3 aurait aussi été sélectionnée si elle avait existé dans le fichier.
Enfin, une troisième forme de condition peut être obtenue en citant les noms de classes de 3ème avec le connecteur .OR. qui signifie "ou". On aurait alors :

```
. list NOM,PRENOM,CLASSE for CLASSE='3TSE' .or. CLASSE='3TSL'
```

Le même type de raisonnement nous permet de proposer deux formes différentes pour la condition e) :

```
. list NOM for .not. '/' $ TELEPHONE
Enreg. N°  NOM
     5     RIVIELLARO
     7     DELVAUX
     9     LUCAS
. list NOM for  '    ' $ TELEPHONE
```

Enfin, la condition f) se décompose en deux conditions simples qui doivent être vraies en même temps : elles seront donc liées par le connecteur .AND. qui signifie bien entendu "et".
La commande sera donc :

```
. list NOM, PRENOM, CLASSE for SEXE = 'F' .and. CLASSE = '1A'
Enreg.   N°    NOM                PRENOM          CLASSE
         8     JACQUART           ANNE-CHRISTINE  1A
```

On peut donc résumer en constatant que dBASE accepte des conditions assez complexes pour autant qu'elles soient composées de CONDITIONS SIMPLES liées entre elles par des CONNECTEURS .NOT., .AND. et .OR. Chaque condition simple comporte un OPERATEUR (=, <>, $ pour les chaînes et =, <, >, <=, >=, <> pour les nombres) et deux expressions où intervient nécessairement un nom de champ.
Différentes fonctions peuvent être employées, SUBSTR n'en est qu'un exemple.

Exercices

Ecrire les commandes dBASE qui permettent de sélectionner les noms et prénoms des élèves qui ...

a) ont l'anglais pour seconde langue;
b) sont enfant unique;
c) possèdent un numéro de téléphone se terminant par 3;
d) ont un responsable légal qui se prénomme ALBERT;
e) redoublent leur 2ème année (Attention ! Ne pas oublier les points pour .T. et .F.);
f) habitant une avenue hors de Namur;
g) ont choisi l'anglais ou l'allemand comme seconde langue.

6. Sélection avec "While"

On vient de voir que le complément FOR permet de sélectionner tous les enregistrements POUR lesquels la condition est vérifiée. dBASE offre une variante avec le complément WHILE qui sélectionne des enregistrements TANT QUE ceux-ci répondent à la condition spécifiée. Ce mode de sélection est d'un usage moins causant mais peut dans certains cas accélérer le travail.

Supposons ainsi que notre fichier élèves compte 1200 enregistrements et que ces derniers soient triés suivant le nom de la classe (par un des moyens que nous étudierons ultérieurement). De ce fait, les fiches d'une même classe sont rassemblées et se suivent : celles de la 3TSE se trouvent, par exemple, entre les numéros 742 et 768.
Avec la commande :

```
. list NOM, PRENOM for CLASSE = '3TSE'
```

dBASE passera en revue les 1200 fiches, pour extraire les 26 demandées cela prendra donc un certain temps (quelques minutes avec un disque dur). On peut accélérer considérablement la manoeuvre en se positionnant directement sur l'enregistrement 742 par un GOTO (*), puis en faisant lister les fiches TANT QUE la classe est 3TSE. La séquence de commandes serait donc :

```
. goto 742
. list NOM, PRENOM while CLASSE = '3TSE'
```

dBASE ne feuilletera ainsi que les 26 fiches demandées.

(*) Nous étudierons bientôt un moyen plus efficace.

Signalons enfin que une clause de portée peut être combinée avec une clause de sélection. Pour demander la liste des doubleurs parmi les 5 fiches suivantes; on donne, par exemple :

```
. list next 5 NOM, PRENOM for DOUBLEUR = .T.
```

7. Affichage des champs MEMO

On se souviendra que la structure de notre fichier comprend un champ COMMENTAIR de type MEMO et est destiné à recevoir toute information complémentaire que l'on pourrait juger utile et qui ne "rentre" pas dans un des autres champs. Ce sera notamment le cas des informations ayant trait à la situation familiale ou à l'état de santé de l'élève.

On peut, comme pour les autres champs, souhaiter dresser une liste de ses informations afin, par exemple, de les communiquer au titulaire de la classe.

Si l'on essaye d'obtenir tous les champs de chaque enregistrement en tapant :

```
. list all
```

on sera déçu de constater que sous le titre COMMENTAIR, dBASE ne nous renvoie que le mot "memo". En fait, la commande LIST n'affiche le texte associé à une zone mémo que si le nom du champ concerné est cité nommément dans la liste des "FIELDS". Ce texte est alors affiché sur une largeur de 50 caractères :

```
. list off NOM, COMMENTAIR
NOM                    COMMENTAIR
LECOMTE
LACROIX
GYSEL
BRASSEUR               Problèmes familiaux :
                       - Décès accidentel des parents en 1984.
                       - Confié à la garde de son oncle.
RIVIELLARO
DEFLEUR
DELVAUX
JACQUART               Problèmes de santé :
                       - Vue diminuée       --> A placer au 1er rang.
LUCAS
```

Comme d'habitude, il est possible d'ajouter une clause de sélection mais celle-ci ne peut jamais porter sur le contenu du champ mémo. Ainsi,

```
. list off NOM, COMMENTAIR for CLASSE = '1A'
```

est tout à fait valide mais par contre :

```
. list off NOM, COMMENTAIR for 'santé' $ COMMENTAIR
```

ne sera pas accepté par dBASE.

8. Affichage de "combinaisons" de champs

La présentation des listes que nous avons obtenues jusqu'à présent peut paraître assez monotone et parfois peu fonctionnelle. Vous pourriez ainsi vouloir rassembler le NOM et le PRENOM de l'élève en une seule expression, séparés par un seul blanc. La commande LIST, autorise ce genre d'affichage : il suffit de placer dans sa liste de champ (complément FIELDS) une expression décrivant le résultat

souhaité.

Une fois de plus, nous ferons appel à des outils (fonctions) pour énoncer cette expression. Parmi ceux-ci, la fonction TRIM est certes, la plus employée. Elle a pour effet d'enlever les caractères blancs qui se trouvent à la fin d'une chaîne. Ainsi,

trim ('ALBERT ') produit la chaîne 'ALBERT'

Cette fonction utilisée conjointement à l'opération + qui met bout à bout deux chaînes de caractères pour n'en plus faire qu'une, va nous permettre de réaliser des tas de combinaisons.
Ainsi,

'BON' + 'JOUR' produit la chaîne 'BONJOUR'

Pour vous familiariser avec ces deux outils nouveaux, essayer les quelques exercices suivants :

```
. goto 2
.
. ? NOM
LACROIX
.
. ? NOM+PRENOM
LACROIX    MARIE
.
. ? trim(NOM)+PRENOM
LACROIXMARIE
.
. ? trim(NOM)+' '+PRENOM
LACROIX MARIE
```

Cette dernière expression est celle que nous utiliserons dans la commande LIST qui devient ainsi :

```
. list trim(NOM)+' '+PRENOM, TELEPHONE
Enreg.  N°    trim(NOM)+' '+PRENOM                TELEPHONE
      1       LECOMTE CECILE                      081/363432
      2       LACROIX MARIE                       081/335637
      3       GYSEL MARC-ALEXANDRE                081/223344
      4       BRASSEUR ALAIN                      081/434413
      5       RIVIELLARO SAMINA
      6       DEFLEUR ERIC                        081/734598
      7       DELVAUX LAURA
      8       JACQUART ANNE-CHRISTINE             081/304023
      9       LUCAS BENOIT
```

De très nombreuses combinaisons sont permises. En voici, par exemple, une utilisant SUBSTR. A vous de décrypter l'expression utilisée !

```
. list trim(NOM)+' '+substr(PRENOM,1,1)+'.', NAISSANCE
for CLASSE = '2CO'
Enreg. N°    trim(NOM)+' '+substr(PRENOM,1,1)+ '.'  NAISSANCE
      2      LACROIX M.                             03/12/75
      6      DEFLEUR E.                             24/07/75
      9      LUCAS B.                               09/04/74
```

9. Affichage sur imprimante

La plupart des listes que nous venons de montrer ne sont pas d'une grande utilité si l'on ne peut en garder une trace concrète. Ainsi, pour obtenir une copie de la liste sur l'imprimante, il suffit de terminer la commande LIST par le complément TO PRINT comme dans l'exemple suivant :

```
. list NOM, PRENOM, ADRESSE to print
```

10. Emploi de la commande DISPLAY

Le logiciel dBASE dispose d'une seconde commande pour l'affichage du contenu d'un fichier. Il s'agit de DISPLAY qui accepte exactement les mêmes compléments que LIST. DISPLAY ne se différencie de son sosie que pour de petits détails :

- Par défaut, DISPLAY n'affiche que le contenu de la fiche courante; pour obtenir tout le fichier, on doit demander DISPLAY ALL ...
- Si l'affichage des données demandées abouti à remplir plus d'un écran, DISPLAY fait une pause et affiche le message

  "Appuyer sur une touche pour continuer ..."

  L'examen, à l'écran, d'une liste assez longue s'en trouve ainsi facilitée (*).

A ces deux remarques près, les commandes LIST et DISPLAY peuvent être considérées comme équivalentes. Aussi, dans ces notes, nous n'emploierons plus que LIST.

(*) Avec LIST, on peut aussi stopper le défilement et le relancer après en pressant ^S.

V. MODIFICATION DES DONNEES

Une des raisons qui pousse l'utilisateur potentiel à informatiser la gestion d'une base de données, c'est de pouvoir en assurer la mise à jour de façon simple et rapide afin qu'elle soit constamment le reflet de la situation réelle qu'elle représente.

Nous allons à présent étudier les moyens que dBASE nous offre pour modifier les données de nos fichiers.

1. Modification d'une fiche particulière

L'élève Eric Defleur vient de déménager. Pour mettre son adresse à jour, nous allons tout d'abord nous positionner sur sa fiche avec la commande GOTO (*), puis nous en demanderons l'édition :

```
. goto 6
. edit
```

dBASE affiche alors la fiche complète de l'élève en reprenant la même disposition que lors de l'introduction des données (commande APPEND). Il ne nous reste qu'à déplacer le curseur, jusqu'aux champs à modifier, en employant les touches flèches. Le cadre situé dans le haut de l'écran nous rappelle, le cas échéant les combinaisons de touches admises à ce niveau. Notons particulièrement ^Y qui efface d'un seul coup le contenu d'un champ et les habituels ^Home pour accéder à la zone mémo, ainsi que ^End pour quitter en sauvegardant les modifications.

Voici par exemple le contenu modifié de la fiche.

(*) D'autres moyens seront étudiés au chapitre suivant.

```
NOM          DEFLEUR
PRENOM       ERIC
RESP_LEGAL   DEFLEUR ALBERT
ADRESSE      Place du Marché aux Légumes, 2
CODEPOSTAL   5000
LOCALITE     NAMUR
TELEPHONE    081/229356
NAISSANCE    24/07/75
SEXE         M
FRERES       1
SEC_LANG     N
CLASSE       2CO
DOUBLEUR     T
COMMENTAIR   memo
```

La commande EDIT nous permet aussi de "feuilleter" le fichier. On passe en effet à la fiche antérieure en pressant la touche PgUp ou à la fiche suivante avec la touche PgDn.

2. Modification d'un même champ de plusieurs fiches

Au début de l'année scolaire, une modification importante du fichier élèves consiste à changer le contenu du champ CLASSE et éventuellement du champ DOUBLEUR. Nous pouvons bien entendu employer EDIT pour passer tous les enregistrements en revue mais cela sera long et fastidieux car nous devrons "survoler" de nombreux champs dont le contenu ne doit pas être modifié.

L'idéal est de pouvoir sélectionner les fiches classe par classe et de n'éditer que les champs à modifier. Cela est possible avec la commande CHANGE qui est un sosie amélioré de EDIT.
En effet, si l'on demande simplement :

```
. change
```

on obtient le même écran qu'avec EDIT.

Par contre, CHANGE est capable de reconnaître le complément FIELDS (ce mot est ici obligatoire) ainsi que les clauses de portée (RECORD ..., NEXT ...) et de condition (FOR ... ou WHILE ...) .

En nous inspirant de ce que nous faisions avec LIST, on peut donc écrire :

```
. change fields NOM,PRENOM,CLASSE,DOUBLEUR for CLASSE='2CO'
```

Les champs NOM et PRENOM sont indispensables pour pouvoir identifier clairement la fiche que l'on modifie. Cependant, comme ces deux champs ne doivent pas subir de mise à jour, il est judicieux de les faire apparaître après les champs à modifier. Le changement de fiche par PgDn ou PgUp ramène en effet le curseur au début du premier champ édité. La commande est donc :

```
. change fields CLASSE,DOUBLEUR,NOM,PRENOM for CLASSE='2CO'
```

et provoque l'apparition d'une "mini-fiche" pour le premier enregistrement concerné (n°2) :

```
CLASSE      2CO
DOUBLEUR    F
NOM         LACROIX
PRENOM      MARIE
```

De plus, une pression sur la touche PgDn nous amène bien sur la seconde fiche qui nous intéresse (n° 6) et ainsi de suite.

3. Modification en mode "Table"

La table est, nous l'avons montré dans le premier fascicule, la représentation classique, et fort naturelle d'ailleurs, d'un fichier appartenant à une base de données. La commande LIST nous a aussi habitué à cette présentation.

Il est donc naturel d'espérer pouvoir modifier les données lorsqu'elles sont présentées sous forme de table.

La commande BROWSE affiche ainsi 17 enregistrements sur l'écran sous forme de table. Le nombre de colonnes affectables simultanément est évidemment limité par la largeur de l'écran (80 caractères).

Avec notre fichier élèves, la commande :

```
. goto 1
. browse
```

nous amène l'écran suivant (surmonté de l'habituel résumé des touches).

NOM----------	PRENOM----------	RESP_LEGAL----------
LECOMTE	CECILE	LECOMTE MARCEL
LACROIX	MARIE	LACROIX JEAN-PAUL
GYSEL	MARC-ALEXANDRE	GYSEL RENEE
BRASSEUR	ALAIN	DASSY EMILE
RIVIELLARO	SAMINA	RIVIELLARO RENZO
DELFEUR	ERIC	DEFLEUR ALBERT
DELVAUX	LAURA	DELVAUX YVES
JACQUART	ANNE-CHRISTINE	JACQUART LOUIS
LUCAS	BENOIT	MAHY BERNADETTE

où les champs de la fiche courante (la première dans ce cas) sont mis en inversion vidéo. Les touches "flèches" haut et bas permettent de se déplacer d'une fiche à l'autre. Pour voir les champs suivants (ADRESSE,CODEPOSTAL,...), on doit déplacer la "fenêtre" en pressant ^-> (CTRL et flèche droite) ou ^<-. On voit ainsi d'autres colonnes :

RESP_LEGAL	ADRESSE	CODEPOSTAL
LECOMTE MARCEL	Rue de la Station, 73	5730
LACROIX JEAN-PAUL	Rue de la Libération, 2a	5002
GYSEL RENEE	Place du Petit Pont, 3, bte 2	5000
DASSY EMILE	Avenue de l'Abbaye, 45	5750
RIVIELLARO RENZO	Rue des Vignerons, 12	5000
DEFLEUR ALBERT	Place du Marché aux Légumes, 2	5000
DELVAUX YVES	Chaussée des Pavés, 5	5730
JACQUART LOUIS	Cité des Rossignols, 30	5100
MAHY BERNADETTE	Avenue Cardinal Mercier, 17	5000

Pour modifier un champ, il suffit de déplacer le curseur à l'intérieur de cette zone, puis de taper les mises à jour. Les modes insertion et recouvrement sont tous deux utilisables, on sélectionne le mode désiré avec la touche Ins.

Revenons à notre problème : nous voudrions obtenir une table ne reprenant que les champs NOM, PRENOM, CLASSE et DOUBLEUR. Et bien, il suffit ici aussi de spécifier un complément FIELDS :

```
. browse fields NOM,PRENOM,CLASSE,DOUBLEUR
```

pour obtenir la table :

NOM	PRENOM	CLASSE	DOUBLEUR
LECOMTE	CECILE	3TSE	F
LACROIX	MARIE	2CO	F
GYSEL	MARC-ALEXANDRE	3TSL	F
BRASSEUR	ALAIN	1A	F
RIVIELLARO	SAMINA	3TSE	T
DEFLEUR	ERIC	2CO	T
DELVAUX	LAURA	3TSL	F
JACQUART	ANNE-CHRISTINE	1A	F
LUCAS	BENOIT	2CO	T

que nous pouvons facilement modifier.

Il est un peu regrettable que dBASE n'ait pas prévu, ici, la possibilité d'ajouter une clause de condition du type FOR. Cet inconvénient pourra cependant être tourné en employant un "filtre" comme nous l'expliquerons bientôt.

Remarques

1. La commande BROWSE accepte quelques compléments qui lui sont particuliers entre autre :
   - LOCK suivi d'un nombre, permet de "figer" quelques colonnes dans le bord gauche de l'écran lors des déplacements de la fenêtre

     ```
     . BROWSE fields NOM,PRENOM,ADRESSE,CODEPOSTAL,LOCALITE ;
               lock 1
     ```

     bloquera le 1er champ (NOM) et permettra ainsi de toujours savoir à quel nom font référence l'adresse, le code postal et la localité.
   - FREEZE suivi d'un (seul) nom de champ spécifie la zone pour laquelle on va accepter des modifications. Les autres champs seront affichés mais ne pourront être modifiés. (*)

2. En pressant ^Home, on accède à un menu, en haut de l'écran :

   ```
   Fin          Début          Figer          Enreg. N°          Geler
   ```

   En sélectionnant le bon mot avec les flèches, puis en pressant RETURN, on peut :

   Fin : se positionner à la fin du fichier

   Début : se positionner au début du fichier

(*) Le terme "Freeze" qui signifie bien entendu "geler" est vraiment mal choisi car il s'agit justement de la seule rubrique qui ne soit pas "prise dans la glace" !

Enreg. N° : se positionner sur l'enregistrement dont on donnera le numéro (équivalent à une commande GOTO)

Figer : sélectionner le nombre de champ à figer (comme LOCK)

Geler : désigner le champ à geler (comme FREEZE)

4. Modification systématique d'un champ

Imaginons encore un autre genre de mise à jour du fichier. Nous allons d'abord ajouter un champ SOLDE à notre fichier pour comptabiliser, par année scolaire, le montant dû par les parents, pour les frais divers (location de livres, petit matériel, excursion, ...). A vous d'employer la commande MODIFY STRUCTURE pour ajouter ce nouveau champ (entre DOUBLEUR et COMMENTAIRE). Le type de la zone SOLDE sera bien entendu numérique et aura une dimension de 4 chiffres (pas de décimales).

La modification au jour à jour pourra se faire à l'aide des commandes EDIT, CHANGE ou BROWSE. Par contre, au 1er septembre de chaque année nous devrons remettre tous les compteurs à zéro. Réaliser cette tâche avec les commandes citées ci-dessus serait extrêmement fastidieux. Nous emploierons plutôt la commande REPLACE de la façon suivante :

```
. replace all SOLDE with 0
    9 enreg. remplacé(s)
```

La clause de portée ALL est indispensable car, par défaut, la commande REPLACE n'agit que sur la fiche courante. dBASE nous indique, après exécution un nombre de remplacements effectués.

Voyons, à présent, comment nous acquitter de la tâche suivante. Les élèves de '1A' sont tous partis en excursion et il convient d'ajouter 250 frs au SOLDE qu'ils doivent payer.

Nous devrons donc d'abord sélectionner les élèves de 1A. Pour cela, la clause de condition FOR (éventuellement WHILE) pourra être utilisée. Ensuite, il faudra, pour chacun des élèves, remplacer la somme qu'il devait déjà par une nouvelle somme obtenue en ajoutant 250 à cette ancienne somme. Si vous avez déjà utilisé un langage de programmation comme PASCAL ou BASIC vous reconnaîtrez là une classique action d'affectation que l'on peut énoncer avec la phrase :

"place SOLDE+250 dans SOLDE"

L'expression SOLDE+250 y désigne une information dont la valeur "concrète" sera déterminée, en temps opportun, par l'ordinateur en réalisant une addition entre ce qu'il trouvera alors dans le champ SOLDE et la constante 250.
La commande complète s'énonce donc :

```
. replace all SOLDE with SOLDE+250 for CLASSE = '1A'
        2 enregs. remplacé(s)
```

On peut vérifier que seuls, les 2 enregistrements sélectionnés ont été modifiés en listant les champs NOM, CLASSE et SOLDE :

```
. list off NOM, CLASSE, SOLDE
NOM                  CLASSE       SOLDE
LECOMTE              3TSE             0
LACROIX              2CO              0
GYSEL                3TSL             0
BRASSEUR             1A             250
RIVIELLARO           3TSE             0
DEFLEUR              2CO              0
DELVAUX              3TSL             0
JACQUART             1A             250
LUCAS                2CO              0
```

Cette commande REPLACE est très puissante car on peut utiliser de nombreuses opérations et fonctions pour "calculer" la nouvelle valeur d'une rubrique. Signalons à ce propos que dBASE reconnaît les opérateurs mathématiques suivants :

+ symbole de l'addition
- symbole de la soustraction
* symbole de la multiplication
/ symbole de la division

ainsi que diverses fonctions dont notamment :

INT (X)	qui fournit la partie entière du nombre X
ROUND (X, d)	qui fournit l'arrondi du nombre X en conservant des décimales
SQRT (X)	qui fournit la racine carrée de X

On peut donc imaginer (*) une commande REPLACE pour diminuer de 5 % les soldes dus de tous les élèves. Ce serait par exemple :

```
. replace all SOLDE with ROUND(SOLDE *0.95 , 0)
```

(*) C'est hélas fort improbable dans la réalité !

VI. SUPPRESSION DE FICHES

Au cours de la vie d'un fichier, on est amené à ajouter de nouveaux enregistrements et à modifier le contenu de fiches existantes. Tôt ou tard, il arrive que des fiches deviennent inutiles. Ce sera le cas pour nous lorsqu'un élève quittera l'école.

dBASE a bien entendu prévu le moyen de supprimer des enregistrements du fichier. Cependant, pour des raisons de sécurité (ainsi que de rapidité d'exécution) la suppression d'enregistrements se réalise en deux temps.

Le premier temps consiste à "endormir" la fiche. Elle est aussi "marquée pour effacement" mais reste physiquement présente dans le fichier. Pour cette raison, on parle aussi d'effacement logique de fiches. Lorsqu'une fiche est "endormie", il reste toujours possible de la "réveiller"; elle reprend alors sa place dans la base de données comme si elle n'avait jamais été menacée.

Si l'on désire vraiment voir disparaître les fiches "endormies" présentes dans le fichier, on donne la commande PACK. Le logiciel recopie alors tout le fichier sur lui-même en omettant les fiches précédemment marquées. Cette opération n'est donc plus réversible, les fiches supprimées ne pourront plus être retrouvées par dBASE.

1. Comment endormir une fiche ?

Il existe plusieurs procédés pour endormir des fiches. La première, la plus directe consiste à utiliser la commande DELETE qui accepte les habituelles clauses de portée et de condition.
Ainsi, pour endormir la fiche numéro 4, on donne la commande :

```
. delete record 4
  1 enreg. supprimé(s)
```

Si l'on demande alors une liste des noms, on obtient :

```
. list NOM
Enreg.  N°       NOM
        1        LECOMTE
        2        LACROIX
        3        GYSEL
        4  *     BRASSEUR
        5        RIVIELLARO
        6        DELFEUR
        7        DELVAUX
        8        JACQUART
        9        LUCAS
```

On voit que l'enregistrement 4 apparaît toujours mais est précédé d'une petite étoile rappelant son sommeil. Si l'on désire que les enregistrements endormis n'apparaissent plus lors des LIST et autres DISPLAY, on utilisera la commande SET DELETED ON. A partir de ce moment, la base de données se comportera vraiment comme si les enregistrements endormis avaient disparu comme le montre la séquence suivante :

```
. set deleted on
. list NOM
Enreg.  N°       NOM
        1        LECOMTE
        2        LACROIX
        3        GYSEL
        5        RIVIELLARO
        6        DEFLEUR
        7        DELVAUX
        8        JACQUART
        9        LUCAS

. list NOM, PRENOM, for NOM='BRASSEUR'
Enreg.  N°       NOM                  PRENOM
```

Seule la spécification du complément RECORD ... permet de montrer l'enregistrement endormi.

```
. list NOM record 4
Enreg.  N°      NOM
        4  *    BRASSEUR
```

Pour revenir en arrière et retrouver l'accès aux enregistrements en sommeil, on utilise :

```
. set deleted off
```

Ceux-ci restent néanmoins endormis (l'étoile qui les marque en est la preuve), pour les réveiller tout à fait, c'est la commande RECALL qui sera employée. Elle admet les mêmes règles de syntaxe que DELETE. Pour tout récupérer, on peut donc taper :

```
. set deleted off
. recall all
  1 enreg. restitué(s)
.
```

A côté de la commande directe DELETE, il est également possible d'endormir des fiches pendant l'emploi des commandes de mise à jour EDIT, CHANGE et BROWSE. Toutes les trois prévoient en effet la combinaison de touches ^U pour "détruire" l'enregistrement en cours d'édition.
La fiche n'est dans ce cas pas vraiment détruite mais est seulement mise en sommeil. On peut d'ailleurs la réveiller en pressant une seconde fois ^U. L'état endormi (ou éveillé) est confirmé visuellement par la présence (ou l'absence) dans le haut de l'écran du message *DET*.

2. Suppresion définitive des fiches endormies

Il est temps, à présent, d'essayer la commande de suppression définitive. Néanmoins, pour ne pas perdre le fichier ELEVES que nous avons créé et mis à jour jusqu'ici nous allons commencer par demander à dBASE d'en exécuter une copie complète sous le nom ELEVEBIS. La commande de copie est la suivante (*) :

```
. copy to ELEVEBIS
  9 enreg. copié(s)
```

Nous pouvons maintenant effacer des enregistrements sans crainte. Supposons donc que les fiches des élèves de '2CO' doivent être éliminés.

Pour les endormir, nous emploierons la commande :

```
. delete all for CLASSE = '2CO'
        3 enregs. supprimé(s)
```

Vérifions par un LIST.

```
. list NOM, CLASSE
Enreg.  N°      NOM              CLASSE
        1       LECOMTE          3TSE
        2 *     LACROIX          2CO
        3       GYSEL            3TSL
        4       BRASSEUR         1A
        5       RIVIELLARO       3TSE
        6 *     DEFLEUR          2CO
        7       DELVAUX          3TSL
        8       JACQUART         1A
        9 *     LUCAS            2CO
```

(*) Nous étudierons cette commande en détail un peu plus loin.

Demandons la suppresion définitive :

```
. pack
        6 enregs. copié(s)

. list NOM, CLASSE
Enreg.  N°      NOM             CLASSE
        1       LECOMTE         3TSE
        2       GYSEL           3TSL
        3       BRASSEUR        1A
        4       RIVIELLARO      3TSE
        5       DELVAUX         3TSL
        6       JACQUART        1A
```

Le listing obtenu cette fois-ci, nous montre bien que les trois enregistrements sont définitivement disparus. Les fiches restantes ont ainsi été rénumérotées de façon consécutive.

Remarques

1. Si l'on doit supprimer tous les enregistrements d'un fichier, il existe une commande de "mise à blanc" mais elle doit être utilisée avec beaucoup de discernement car elle ne permet pas de revenir en arrière. Il s'agit de ZAP.

2. Pour retrouver votre fichier élèves, tel qu'il était avant suppression des trois fiches, il suffit d'exécuter la manoeuvre de copie dans le sens contraire :

```
. use ELEVEBIS
. copy to ELEVES
ELEVES.dbf existe déjà, voulez-vous l'écraser ? (O/N) Oui
  9 enregs. copié(s)
. use ELEVES
```

BIBLIOGRAPHIE

[1] COHEN R.

dBASE III Plus en mode direct.

PSI, Paris, 1987.

[2] COHEN R.

Programmer en dBASE III et III Plus.

PSI, Paris, 1986.

[3] FRASSON C.

Utiliser les bases de données avec dBASEII et dBASEIII.

Les Editions d'Organisation, Paris, 1986.

[4] HERGERT D.

Dictionnaire dBASE III

Cedic / Nathan, Paris, 1986.

[5] KELLER M.

Clefs pour dBASE III Plus.

PSI, Paris, 1987.

[6] LILEN H.

Pratique de dBASE III Plus.

Editions Radio, Paris, 1987.

[7] MICHEL C.

Ecrire en dBASE.

BCM S.C., Banneux, 1986.

[8] SIMPSON A.

Introduction à dBASE III.

Sybex, Paris, 1985.

Annexe 1

Enregistrement nº 1 : LECOMTE CECILE
Parents : LECOMTE MARCEL
Rue de la Station, 73
5730 MALONNE
Téléphone : 081/363432
Date de naissance : 23/05/74 Sexe : F
Frères et soeurs : 2 Sec. Langue : N
Classe : 3TSE Doubleur : Non
Commentaire :

Enregistrement nº 2 : LACROIX MARIE
Parents : LACROIX JEAN-PAUL
Rue de la Libération, 2a
5002 SAINT-SERVAIS
Téléphone : 081/335637
Date de naissance : 03/12/75 Sexe : F
Frères et soeurs : 0 Sec. Langue : N
Classe : 2CO Doubleur : Non
Commentaire :

Enregistrement nº 3 : GYSEL MARC-ALEXANDRE
Parents : GYSEL RENEE
Place du Petit Pont, 3, bte 2
5000 NAMUR
Téléphone : 081/223344
Date de naissance : 17/06/74 Sexe : M
Frères et soeurs : 1 Sec. Langue : L
Classe : 3TSL Doubleur : Non
Commentaire :

Enregistrement nº 4 : BRASSEUR ALAIN
Parents : DASSY EMILE
Avenue de l'Abbaye, 45
5750 FLOREFFE
Téléphone : 081/434413
Date de naissance : 29/02/76 Sexe : M
Frères et soeurs : 1 Sec. Langue : A
Classe : 1A Doubleur : Non
Commentaire :
Problèmes familiaux :
- Décès accidentel des parents en 1984.
- Confié à la garde de son oncle.

Enregistrement nº 5 : RIVIELLARO SAMINA
Parents : RIVIELLARO RENZO
Rue des Vignerons, 12
5000 NAMUR
Téléphone :
Date de naissance : 07/03/74 Sexe : F
Frères et soeurs : 4 Sec. Langue : A
Classe : 3TSE Doubleur : Oui
Commentaire :

3

FORMATION EN EDUCATION N°1.1

RECHERCHE

CENTRE O.S.E.

Grille d'analyse de didacticiels.

Jean DONNAY

Marc ROMAINVILLE

DEPARTEMENT
EDUCATION & TECHNOLOGIE

Facultés Universitaires Notre-Dame
de la Paix B-5000 Namur

RÉSEAU O.S.E.

Ordinateur au Service de l'Éducation

GRILLE D'ANALYSE DE DIDACTICIELS

J. DONNAY

M. ROMAINVILLE

version provisoire (juin 84)

CENTRE O.S.E. NAMUR
Facultés Universitaires N.D. de la Paix
Unité de Pédagogie
61, rue de Bruxelles - 5000 NAMUR

INTRODUCTION

Le but de ce document est d'offrir aux enseignants une synthèse des questions essentielles qu'il convient de se poser avant de choisir un DIDACTICIEL. (1)

Cette grille d'analyse est donc un instrument d'évaluation de produits éducatifs informatiques et peut également servir de base de réflexion pour la construction d'un didacticiel. Il serait aussi possible de constituer à l'aide de cette grille un répertoire critique des didacticiels disponibles.

Quels critères ?

Sur quelles bases juger un didacticiel ? Comment déterminer les caractéristiques d'un "bon" didacticiel ? Quels critères utiliser ? Etablir une grille d'évaluation de produits éducatifs implique la détermination des points de vue considérés.

Or, selon nous, toute évaluation de produit éducatif -et même plus largement de stratégie d'enseignement- requiert avant tout de la part de l'évaluateur une connaissance des mécanismes d'apprentissage. Notre approche de l'analyse de l'enseignement est centrée sur l'apprenant et sur les conditions d'apprentissage. Nous examinerons l'adéquation entre, d'une part, les caractéristiques du produit et, d'autre part, ce que l'on sait des processus d'apprentissage et des conditions d'environnement qui y sont favorables.

Chaque rubrique de la grille fait donc référence à une ou à plusieurs caractéristiques des processus par lesquels l'apprenant construit ses connaissances.

Ainsi, on sait actuellement que l'information est stockée dans notre cerveau sous forme de réseau non linéaire. De plus, l'information nouvelle doit s'intégrer dans un réseau préexistant. Ces données issues de la psychologie cognitive permettent déjà d'énoncer une série de critères d'évaluation des didacticiels : quel est le degré de souplesse du didacticiel ? Permet-il des allées et venues dans la matière en fonction de l'apprenant ? Tient-on compte de son état de départ ? Comment l'information présentée est-elle structurée ? La mise en évidence des mots, phrases, paragraphes est-elle réalisée ? Quelle est la marge de prise en compte des souhaits de l'apprenant ? etc...

Il nous reste à présenter les critères retenus -et les théories sousjacentes- : ils se répartissent en deux groupes correspondant aux deux questions suivantes :

a. Qu'est-ce qu'apprendre ?
b. Quelles sont les conditions d'environnement favorisant l'apprentissage ?

Nous avons également ajouté une troisième catégorie concernant le contenu.

Pour la facilité de lecture, un signet reprend les 15 critères principaux. A chaque rubrique, vous trouverez dans la marge de gauche le numéro du critère qui a suscité la question d'évaluation présentée.

Théories sousjacentes à nos critères

a. De manière générale, tout apprentissage implique un CHANGEMENT DURABLE DANS LES STRUCTURES COGNITIVES de l'apprenant.

a.1. Nous parlons de "structures cognitives" à dessein car il est

(1) On appelle "didacticiel" tout programme informatique utilisé à des fins d'enseignement.

désormais clair que le stockage d'information se réalise, dans notre cerveau, sous forme de réseau non linéaire.

L'information, bien que traitée linéairement par les organes d'entrée et de sortie (l'ouïe et la parole) est stockée sous forme de réseau cognitif. Pour s'en convaincre intuitivement, il suffit de penser au réseau sémantique tissé autour d'un concept, d'un nom qui vous est cher, d'un moment. L'ordinateur a, de ce point de vue, des possibilités dont on commence à peine à tirer profit : l'information peut être présentée conjointement par l'image et par le son, l'apprenant peut potentiellement avancer, reculer dans le cours en suivant le cheminement de la construction de son réseau ; il peut en sortir et voir dans quelle partie il se trouve, en demandant les liens avec la suivante, etc...

a.2. Ce changement de structure est une construction active de l'apprenant : il s'agit d'un processus actif individuel et non d'une simple présentation de stimuli. L'apprentissage exige une participation active de l'apprenant, celui-ci doit réorganiser les informations présentées, pouvoir EMETTRE les réponses plutôt que de les écouter ou de les voir. Si ses réponses sont des hypothèses, il faut qu'il puisse les vérifier pour se corriger au besoin.

a.3. Cette construction est individuelle : personne ne peut apprendre à la place de personne ; la réorganisation des informations aboutit à un réseau propre à l'individu, résultat de son style cognitif, de son expérience antérieure, etc... L'ordinateur est un outil idéal pour construire un produit éducatif adapté aux caractéristiques personnelles des apprenants. Il ne faut, bien sûr, pas oublier que, quand il existe, le dialogue enseignant-apprenant reste -et pour longtemps sans doute- le plus adapté (prise en compte de l'histoire de l'élève dans sa totalité).

a.4. Cette construction est une intégration dans une structure précédente. Il y a toujours une structure de départ (même si elle est fausse et doit être abandonnée ; cette structure est nécessaire mais insuffisante : à elle seule, elle ne permet pas de résoudre le problème posé, elle sert néanmoins de point d'ancrage. Lors de l'apprentissage, l'apprenant sera amené à faire la liaison entre la structure de départ et les connaissances nouvelles. Et souvent, l'apprentissage transformera les structures de départ en profondeur. On parlera alors d' "apprentissage en profondeur", par opposition à l' "apprentissage de surface" dans lequel le nouvel élément vient simplement se juxtaposer aux précédents sans provoquer une révision générale de la structure.

a.5. Apprendre, nous l'avons dit, est un changement. Et tout changement implique une dépense d'énergie. L'apprenant doit accepter le défi, reconnaître qu'il y a "désequilibration", investir la tâche et la matière pour construire une nouvelle "équilibration". C'est la motivation de l'apprenant qui est ici en cause.

a.6. Enfin, ce changement dans les structures cognitives doit être durable : il ne suffit pas que l'élève manifeste le comportement souhaité une fois, il faut que celui-ci soit acquis de façon durable. Pour cela, la stratégie d'enseignement doit prévoir des rappels, des exercices fréquents et divers permettant à l'apprenant de mobiliser à nouveau les structures cognitives qu'il avait construites afin de les considérer et de les réactualiser dans des contextes différents.

(1) Le parallèle avec les résultats récents de la neurobiologie est intéressant : pour J-P. CHANGEUX, "apprendre équivaut à modifier temporellement le système neuronal en créant des circuits particuliers (qui sont appelés des graphes) par stabilisation d'un réseau de neurones ..." (10)

b. Quelles sont les CARACTERISTIQUES DE L'ENVIRONNEMENT EDUCATIF QUI FAVORISENT L'APPRENTISSAGE ?

Certaines composantes de l'environnement sont en effet de nature à augmenter la qualité des stratégies et des méthodes d'enseignement. Ces composantes sont les suivantes :

b.7. Le Feedback. L'apprenant est informé de la qualité de sa réponse. L'apprentissage sera d'autant plus efficace que le feedback suivra directement la réponse de l'apprenant. Celui-ci est particulièrement important dans les premières étapes d'une nouvelle tâche pour éviter des apprentissages erronés. De plus, savoir que "tout va bien" et que l'on progresse est un élément important de motivation.

b.8. La correction. Dire que la réponse de l'apprenant est incorrecte ne suffit pas. Un pas de plus est franchi si l'on précise pour quoi, ou du moins, en quoi il s'est trompé et comment rectifier l'erreur. Un diagnostic précis et une remédiation pertinente sont essentiels.

Ces critères 7 et 8 risquent de "départager" les didacticiels : en effet, l'envoi d'un feedback et d'une correction appropriée repose sur une analyse des réponses de l'élève émises dans son "langage". Ce problème restera encore longtemps une des principales difficultés à résoudre.

b.9. La qualité des informations sur l'apprentissage. "Tout apprentissage porte sur certains éléments et appelle la communication d'indices ou d'indications permettant à l'élève de découvrir ces éléments et de savoir ce qu'il doit en faire". (3) p. 121 .

On connait désormais l'importance des indices fournis à l'apprenant sur ce qu'il doit apprendre, pourquoi il doit réaliser cet apprentissage, comment, etc... Ces indications permettent également de rendre l'apprentissage plus significatif.

En cours d'apprentissage, ces indications sont essentielles pour savoir "où on est" et ce qu'il reste à faire. Bref, pouvoir se situer à chaque instant (surtout chez l'adulte) parait être un critère important.

b.10. Participation. "L'élève doit participer ou s'exercer, explicitement ou implicitement, en se rappelant ou en utilisant les indices, en essayant de donner les réponses appropriées ou de produire les actes, les comportements ou les réponses spécifiées par les indices, jusqu'à ce qu'ils fassent partie de son répertoire." (3) p. 126

Dans quelle mesure l'apprenant est-il partie prenante ? A-t-il le contrôle sur son apprentissage ? Peut-il choisir le contenu, la méthode, le chemin à parcourir, le rythme d'apprentissage ? Le caractère adaptatif du didacticiel est ici en cause.

b.11. La récompense. La plupart des théories de l'apprentissage ont montré le rôle important du renforcement dans l'installation d'un nouveau comportement.

Comment le didacticiel récompense-t-il "intrinséquement" l'apprenant ? Renforce-t-il les "bons" comportements ?

c. CARACTERISTIQUE DU CONTENU

Le point a. est centré sur l'apprenant, le point b. sur l' environnement. Il nous a semblé indispensable de constituer une rubrique supplémentaire centrée sur la matière. Les dimensions envisagées sont les suivantes :

c.12. l'exactitude des contenus présentés à l'apprenant

c.13. leur pertinence par rapport au niveau scolaire, aux objectifs, etc...

c.14. la structuration de la matière, l'organisation des concepts, etc...

c.15. la présentation. Ce dernier critère concerne plus l'aspect formel de la présentation du contenu.

GRILLE D'EVALUATION

Avant la présentation de la grille, 4 remarques générales sur l'évaluation de produits éducatifs informatiques s'imposent.

1. Tout programme, à la première utilisation, peut paraître attrayant. L'utilisation du matériel informatique, les possibilités sonores ou graphiques représentent une stimulation nouvelle souvent motivante. Cependant, une évaluation en profondeur demande

   a) de "maltraiter" le didacticiel : donner de mauvaises réponses, des réponses inappropriées, essayer de tricher, etc...

   b) de faire essayer le programme à des élèves de niveau visé. L'observation des réactions des élèves face au didacticiel permettra d'en juger l'adéquation.

2. Tout programme devra être avant tout évalué en fonction des besoins et des objectifs de l'utilisateur : ainsi un programme d'exercices répétitifs sera jugé inutile par un enseignant qui, centré sur les méthodes inductives, recherche un programme de modélisation (construction par l'apprenant d'un modèle à partir de l'observation de faits répétés).

3. Il est impératif de restituer chaque fois l'utilisation d'un didacticiel dans le contexte particulier où il s'inscrira. En effet, les conditions d'apprentissage (seul ou en groupe), la justification dans une stratégie plus globale (pour quoi faire ?), la justification aux yeux de l'apprenant (c'est important ou non), etc... sont des critères qui dépendent de chaque contexte et peuvent faire varier grandement les effets attendus.

   Par exemple, pour un élève très motivé, nos exigences de disposition des informations sur l'écran sont sans doute superflues. De même si l'enseignant prévient les élèves des défauts du produit et s'il les corrige anticipativement, voilà qui peut combler bien des lacunes.

4. Comme nous le verrons, il existe plusieurs types de didacticiels : le tutoriel, les exercices répétitifs, la simulation, etc... Les critères d'évaluation proposés ne s'appliquent pas tous à tous les types de didacticiels. De plus, la liste des questions n'est ni exhaustive ni contraignante. L'utilisateur peut insérer ou supprimer une question en fonction de ses objectifs, de son style d'enseignement, du contexte particulier dans lequel il va utiliser le didacticiel, etc... (cfr. remarques 2 et 3). Ainsi le point (1.11) sur les réactivations ne concerne que le tutoriel.

Pour faciliter l'utilisation de la grille, nous avons reproduit à droite de chaque critère une échelle d'évaluation à 3 classes. La signification de ces 3 classes est la suivante :

+ : le didacticiel satisfait au critère considéré. Ce trait est à mettre à l'actif du didacticiel.

- : le didacticiel ne satisfait pas au critère considéré.

0 : par commodité, nous avons rassemblé dans cette catégorie, d'une part les cas d'ambiguïté (on ne sait pas se prononcer, le didacticiel

tantôt satisfait au critère et tantôt pas), et d'autre part les cas pour lesquels le critère n'est pas pertinent pour le type de didacticiel considéré.

Le lecteur pourra donc -s'il le désire- obtenir une évaluation globale du didacticiel en comptabilisant les points positifs et négatifs.

Fiches de présentation du didacticiel

CARACTERIQUES GENERALES

Nom du didacticiel : .....

Nom de l'auteur : ..... Editeur : .....

Date de publication : ..... Prix : .....

Public cible : ..... Discipline : .....

Sujet : .....

Objectifs : .....

Prérequis : .....

Durée moyenne d'exécution : .....

Mode d'utilisation : - avec - sans surveillance

- 1 élève - un petit groupe (4) - plus de 4 - peu importe

CARACTERISTIQUES TECHNIQUES

Ordinateur : .....

Système d'exploitation : .....

Support : ..... (si disquette, nbre de drive : .....)

Périphérique d'entrée : ..... (clavier, bâtons commande, etc...)

Périphérique de sortie : ..... (T.V., imprimante)

Utilisation d'autres appareils : .....

Langage programmation : .....

Mémoire vive : .....

La grille est divisée en deux parties :

PARTIE I - Analyse de l'unité de dialogue

L'unité de dialogue est définie comme la plus petite unité pédagogique significative, c'est-à-dire la plus petite unité pédagogique présentable à un apprenant (1). Elle comprend les étapes suivantes (1) (2) :

UNITE D'INTERACTION

1. Envoi d'information à l'apprenant.
2. Envoi d'une sollicitation.
3. Réception d'un message de l'apprenant.
4. Analyse de réponse.

UNITE DE DECISION

(en fonction des informations recueillies et des conditions définies au préalable, l'unité de décision diversifie les réactions du système)

5. Envoi d'un feedback.
6. Branchement vers une autre unité d'interaction.

PARTIE II - Analyse de l'environnement du didacticiel

1. Le type de didacticiel
2. Ses objectifs
3. Son insertion dans le curriculum et dans les activités de la classe
4. La documentation accompagnant le didacticiel
5. Le contrôle du programme
6. Les mesures du rendement du didacticiel
7. L'évaluation des apprentissages
8. Les traces pour l'enseignant.

PARTIE I - ANALYSE DE L'UNITE DE DIALOGUE

UNITE D'INTERACTION

1. Envoi d'informations à l'apprenant

CRITERES	En général...	-	0	+
	- L'information transmise est-elle			
[13]	1. pertinente par rapport aux objectifs ?			
[12]	2. exacte ? (matière, orthographe)			
[4] [13]	3. adaptée au niveau scolaire annoncé ?			
	Idem pour le contexte ?			
	le niveau de langues ?			
	la forme de réponse ?			
	- La lisibilité			
[4] [13]	4. Le vocabulaire utilisé correspond-il au niveau scolaire annoncé ?			
	Pour les jeunes enfants, les spécialistes anglo-saxons de la lisibilité linguistique recommandent d'utiliser en priorité (17) . les mots personnels (le référent est connu de tous) . les mots vivants (les actions, etc...) . les mots précis et concrets de la vie courante. De plus, les mots les plus courts et les mots à fort degré de "visualisation" sont mieux mémorisés.			
[4] [13]	5. La longueur des phrases correspond-elle au niveau scolaire annoncé ?			
	La lisibilité et la mémorisation des mots d'une phrase varie en effet en fonction de la longueur de celle-ci ; surtout chez les jeunes enfants, les phrases courtes ont plus de chances d'être retenues (ex. 8 mots pour des élèves de 3ème primaire). Cependant, la structure des phrases est un autre facteur qui explique la compréhension et la mémorisation.			

[1] [9] 6. La structure des phrases favorise-t-elle la lisibilité ?

Une phrase structurée, même un peu longue, sera mieux retenue qu'une phrase simple plus courte. L'emploi de propositions subordonnées et d'enchassements favorise une mémorisation logique de la phrase, porteuse de signification. C'est ainsi que les phrases à structure énumérative sont les moins bien mémorisées (17).

- Pertinence

7. Le programme présente-t-il l'information d'une manière telle qu'il serait difficile de réaliser le même type de présentation avec un autre support pédagogique (le tableau, l'audiovisuel,...)

8. Le programme présente-t-il l'information avant la question ?

Les recherches montrent que les étudiants ne lisent pas le texte ou le graphe placé en dessous de la question.

[13] 9. Est-elle présentée de manière appropriée ?
(si inf. en chiffres ⟶ tableau, etc...)

- Mémorisation

[6] 10. Le programme revient-il sur les notions importantes pour faciliter la mémorisation ?

[6] 11. Propose-t-il plusieurs réactualisations au cours de la leçon ?

[6] [1] 12. Propose-t-il une synthèse après chaque nouvelle unité de matière ?

[6] 13. S'il s'agit de "tutoriel", le programme respecte-t-il les limites d'attention optimales pour la mémorisation ? C'est-à-dire soit une durée de 20 à 40 min. maximum?

soit aménager des pauses pendant une période plus longue ?

Des études sur l'expérience d'E.A.O. télématique en France ont d'ailleurs montré que, dans un contexte extrascolaire, la plupart des enfants, spontanément, ne dépassaient pas ces limites : les élèves du primaire travaillent par tranche de 5 min. pour des exercices de calcul et de 10 min. pour des problèmes à résoudre. A un niveau d'âge plus élevé, la durée de travail oscille entre 20 et 30 min. (21).

[2] [6] 14. Si le temps est fixé, l'information se déroule-t-elle à une vitesse adéquate ?

- Divers

15. S'il existe des "temps morts" (écran vide pendant le chargement), existent-t-ils des messages d'attente ?

[1] 16. Le didacticiel utilise-t-il d'autres sources extérieures de stimulations pour présenter l'information :
. aides visuelles (dia) ?
. aides auditives ?

a. Le graphique

[9] [14] - Focalise-t-il l'attention sur les éléments essentiels?

[15] - Le graphique est-il visible ?
Sa taille est-elle suffisante ?
Son temps de présentation ?

- Sert-il l'objectif pédagogique ?
(ou a-t- il été choisi pour ses seules qualités propres ?)

[1] - Est-il bien relié au texte pour ne former qu'une seule unité d'enseignement ?

[1] [14] - Est-il correctement situé sur la page écran ?
(éviter coupure de texte, éloignement image-commentaires, etc...)

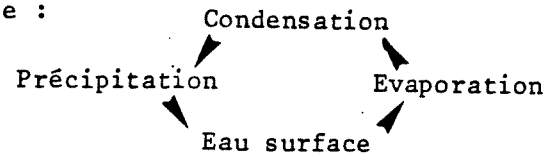
[13] - Le recours à la représentation graphique facilite-t-il réellement la compréhension des données ?

A ce propos, des études anglaises sur l'E.A.O. ont montré que la majorité des élèves de 14-15 ans éprouvaient des difficultés pour interpréter correctement des graphes cartésiens.
Le concepteur de didacticiel doit toujours bien garder à l'esprit que l'interprétation d'un tel graphe exige un transcodage d'une information compacte visuelle en une information verbale ou écrite linéaire et que cette tâche n'est pas évidente pour de jeunes enfants (16).

Si l'interprétation du graphique n'est pas instantanée, des études montrent que ce dernier est alors "ignoré" par l'apprenant qui "perçoit" que le texte fournit une quantité d'informations supérieure dans un temps égal et ne lit que ce dernier.

Une comparaison de lisibilité de diagramme est présentée en annexe 1.

-	0	+

		-	0	+
[15]	- L'image est-elle non-ambigüe ? Plus précisément, il faut veiller à ce que l'apprenant puisse déduire, par lui-même, de l'illustration des informations significatives et correctes. L'excès de schématisation peut nuire à cette opération.			
[14]	- Pour le processus expliqué, le nombre d'illustrations graphiques correspond-il, au moins, au nombre de stades décrits dans ce processus ?			
[1] [11]	- Le didacticiel recourt-il à la représentation chaque fois que cela semble nécessaire ? (schéma, croquis, plan même quand il s'agit de concepts théoriques).			
	Ainsi, dans la résolution de problème, on a constaté qu'une des difficultés d'apprentissage était d'arriver à se construire une "carte mentale" des processus possibles et des directions de solution. De même, la visualisation de certaines notions sous forme de "structure" facilite le processus de mémorisation.			
[14] [9]	- La couleur est-elle utilisée pour porter l'attention sur ce qui est important ? La couleur est un des éléments de la motivation intrinsèque que représente un didacticiel. Les résultats d'études sur la vision permettent de donner les recommandations suivantes aux concepteurs de didacticiel (8) : . éviter les combinaisons incompatibles : rouge/gris, bleu/jaune, vert/bleu et rouge/bleu . accentuer le contraste entre la couleur des caractères et celle du fond d'écran . limiter à 4 le nombre des couleurs pour une image (notamment pour la mémorisation et l'attention) . mettre les caractères alphanumériques en rouge, blanc ou jaune (sur fond noir) . utiliser leurs dénotations habituelles (ex. rouge pour "stop" ou "danger") . limiter l'utilisation du bleu pour les contours, les larges zones périphériques.			
	- Le codage des couleurs est-il cohérent ? (Changement de conventions ?)			
[9]	- Les indices sont-ils utilisés pour porter l'attention sur ce qui est important ? (flèches, soulignement, etc...)			
[4]	- Les mouvements sur l'écran suivent-ils le mouvement habituel des yeux pendant la lecture ?			
[12] [1]	- L'illustration est-elle toujours en accord avec le message ?			
	Exemple :  Le processus décrit est cyclique, une représentation linéaire aurait été moins efficace (13).			

		-	0	+
	Notons encore la possibilité d'associer des images à l'illustration pour favoriser la mémorisation :			
	Précipitation  etc...			
[14] [1]	- Pour des notions nouvelles, les illustrations complexes sont-elles subdivisées ? Dans l'exemple ci-dessus, chaque étape pourrait être présentée isolement avant le schéma global.			
	b. Le texte			
[15]	- Le programme change-t-il de page quand il affiche une information nouvelle ?			
	Libéré des impératifs de l'industrie du papier, le concepteur de didacticiels n'est pas limité en nombre de pages : dès lors, il vaut mieux présenter une idée par page écran.			
[13] [4]	- Le programme évite-t-il le "rouleau" ? (Toutes les informations remontent pour laisser place à une nouvelle information dans le bas de l'écran)			
	Ce système ne correspond pas à la lecture habituelle (gauche-droite et haut-bas), il vaut mieux tourner la page.			
	Qualité de la mise en page			
	La présentation du texte à l'écran vise à assurer, comme pour le livre, une "visualisation des hiérarchies" (Richaudeau) ; il s'agit de mettre en valeur ce qui doit être retenu et plus généralement d'organiser le processus de lecture de l'apprenant.			
[1] [9]	- Les mots importants sont-ils mis en valeur dans le paragraphe ? (par l'utilisation des majuscules, du soulignement, des couleurs, de l'inversion vidéo ou du clignotement) (9)			
	Notons que, contrairement à ce que des études de lisibilité ont montré pour le livre, le public préfère à une très forte majorité lire sur écran un texte rédigé en majuscules. Les tests de mémorisation sont d'ailleurs supérieurs dans ce cas.			
	Des études françaises ont montré que les clignotements nombreux gênent considérablement la lecture ; en effet, l'apparition et la disparation de texte entraîne une fatigue visuelle d'une part, et freine le processus de lecture suivie d'autre part. Les pertes de mémorisation sont alors importantes.			
[1] [9]	- La mise en évidence des paragraphes est-elle bien réalisée ? (en passant une ligne, par une "rentrée"			

	-	0	+

ou un "débord") (12)

D'après les études de SPENCER, c'est cette dernière formule (la première lettre du paragraphe composée en saillie à gauche) qui est la plus efficace pour faciliter la lecture de recherche.

[1] [9] - L'importance des paragraphes est-elle bien mise en évidence ? (encadré ou modifications des zones blanches entre les paragraphes)

Notons que le public préfère des "encadrés classiques", texte et titre centrés, les filets habillant le texte à droite et à gauche, en haut et en bas. Les cadres non fermés amènent le rejet.

[1] [15] - Le texte est-il disposé correctement sur l'écran ? (c'est-à-dire aligné sur la marge de gauche ou éventuellement centré, mais jamais sur la marge de droite).

[15] - Le titre est-il composé correctement ?

On conseille généralement un titre plus large (en double hauteur, double largeur), aligné à gauche ou centré. (cfr. annexe 2)

[1] [9] - les informations complémentaires (accès à d'autres informations, réponses proposées, etc...) sont-elles disposées correctement ?

La meilleure position semble être en bas de l'écran : il n'y a pas de rupture dans la lecture.

(titre ➡ texte ➡ informations complémentaires)

La disposition "verticale" (à gauche ou à droite) du texte entraîne un niveau de rejet assez élevé.

- Le fond d'écran rend-il les textes lisibles ? (dans tous les cas, le fond blanc est à éviter)

2. Envoi d'une sollicitation

Cette sollicitation prend, la plupart du temps, la forme d'une question destinée à l'apprenant : la qualité de la sollicitation dépendra donc directement de la qualité des questions.

Ce n'est pas la quantité de questions posées qui déterminera la valeur du didacticiel, mais plutôt leur qualité : demandent-elles un effort cognitif important de la part de l'apprenant ? Le placent-elles devant une véritable situation-problème ?

Exemple : un automobiliste part de Bruxelles à 11 h. et arrête de rouler à 13 h. Il a fait, en moyenne, du 120 km/h. Ces deux questions semblables impliquent des opérations cogni-

tives toutes différentes :

a) Quelle distance a-t-il parcouru ?
b) Quelle distance a-t-il parcouru sachant que distance = vitesse x temps
= 120 x 2
= ?

- | 0 | +

[2] [10] - Quelle est l'opération cognitive impliquée le plus souvent dans les questions destinées à l'apprenant ?

connaissance ...
compréhension ...
application ...
analyse ...
évaluation ...
synthèse ...

[2] [10] - Qualité de la rédaction des questions ?

. un seul problème par question
. éviter les tournures négatives
. éviter de donner des informations superflues
. pour un Q.C.M.
  - les solutions proposées sont indépendantes l'une de l'autre
  - la B.R. (bonne réponse) ne diffère pas des mauvaises, en forme, longueur et précision
  - les solutions sont allégées et présentées dans un cadre logique.

[9] [10] - L'étudiant sait-il toujours ce qu'il doit répondre ?

Exemple :

```
Préférez-vous A: "Monet"
              B: "Manet"
              C: "Picasso"
              D: "Michel-Ange"
Votre choix ?
```

Doit-il répondre par le nom du peintre préféré, la lettre seule en majuscule, la lettre avec les deux points, etc...

3. Réception du message de l'apprenant

Avec de jeunes enfants surtout, des facilités d'input doivent être prévues : en effet, ceux-ci ont souvent peur que leur réponse ne "casse" l'ordinateur et le peu de facilité d'entrée d'information de la plupart des systèmes renforce cette peur.

[4] - Le didacticiel utilise-t-il les touches les plus simples ? (return, space bar plutôt que CTRL J, etc...)

[4] - Utilise-t-il les codes les plus naturels ?

Exemple : pour une question, les réponses sont souvent "oui - non" et rarement "1 - 0", "0 - N", "Y - N", etc...

- Accepte-t-il un large spectre de réponses ?

[15] - La même fonction est-elle toujours remplie par la même touche ?

Exemple : pour continuer = return, et pas return une fois puis space bar, etc...

[2] [3] - Avant de répondre, l'apprenant peut-il toujours formuler une requête ? (ressources complémentaires, accès à des informations présentées antérieurement, etc...)

[4] - Peut-il taper ses réponses comme il est habitué de le faire ? (ex. : de droite à gauche pour calcul math)

[9] - Le type de réponse que l'on attend de l'élève est-il bien précisé ? (ex. : taper les nombres en chiffres, etc...)

4. Analyse de réponse

-	0	+

Cette rubrique ne concerne que les didacticiels "courageux" qui acceptent les réponses ouvertes. Les questions fermées ne posent pas de problème d'analyse de réponse car, dans ce cas (Vrai Faux (V.F.) ou Question à Choix Multiple (Q.C.M.)), toutes les réponses ont été prévues et sont présentées à l'apprenant, ce dernier doit uniquement choisir celle qui lui semble correcte.

Il est malgré tout conseillé de prévoir une petite analyse de réponse pour les Q.C.M. quand les consignes ne sont pas précisées.

Exemple : la date de la bataille de Waterloo est
1. 1814
2. 1815
3. 1816

Répondez ...

Le didacticiel attend en principe 1, 2 ou 3, mais il devrait pouvoir traiter 1814, 1815, 1816 et même mille huit cent quatorze, etc...

Pour des questions ouvertes, l'apprenant doit construire ses réponses qui seront ensuite analysées.

4.1. Analyse de réponses numériques

Il s'agit ici de reconnaître un nombre accompagné dans certains cas d'une unité.

- Le didacticiel prévoit-il un "arrondi" des nombres ?

  Exemple : considérer que deux valeurs sont identiques à un pourcentage près ; accepter une valeur comprise entre deux valeurs limites, etc...

[3] - Accepte-t-il plusieurs expressions de la même valeur ?

  Exemple : 75/100, 3/4, etc...

[9] - Précise-t-il l'unité souhaitée ?
  Si non, les autres unités correctes sont-elles converties automatiquement et acceptées ?

4.2. Analyse de réponses "texte"

La base de l'analyse est le plus souvent la recherche de mots-clés (ou de "squelette de mots") dans la réponse de l'apprenant (19) (2).

[3] [10] - Le didacticiel prévoit-il des équivalences syntaxiques ? (tolérances des fautes d'orthographe, d'inversions, etc...)

[3] [10] - Le didacticiel prévoit-il des équivalences sémantiques ? (synonymes, etc...)

[3] [10] - le didacticiel peut-il tenir compte de
  . la présence d'un ou plusieurs mot(s)
  . l'absence d'un ou plusieurs mot(s)
  . l'absence et la présence d'un ou plusieurs mot(s)
  . la présence ou l'absence de mots dans un ordre donné
  . la présence ou l'absence de mots dans une liste
  . la présence d'expression exacte.

L'analyse de réponse restera sans doute un des problèmes majeurs de l'E.A.O. Comme le notent H. BESTOUGEFF et J-P. FARGUETTE, la simple question "donnez la relation entre la tension, la résistance et le courant" montre que, si la réponse est donnée sous forme de texte, ce type d'analyse par mots-clés a ses limites. Ils expliquent :

"Par exemple, dans le cas présent, l'apprenant qui répond : "la RESISTANCE est EGALE au QUOTIENT de la TENSION par l'INTENSITE du COURANT", n'est pas compris par la machine (les mots-clés sont dans l'ordre : "tension" puis "égale", puis "produit", puis "résistance et intensité", puis "courant". Ils correspondent à la phrase correcte : la Tension est Egale au Produit de la Résistance par l'Intensité du Courant. Les deux mots "Egale et Courant" ne sont pas obligatoires ; "Résistance et Intensité" peuvent se mettre dans un ordre quelconque après "Produit").

Tandis que le petit farceur qui écrit : "Ma grand-mère à une tension sans égale quand elle produit un effort

- 0 +

d'une intensité bien supérieure à ce que lui permet sa résistance physique", se verra attribuer un commentaire laudatif." ((2) p. 140)

-	0	+

UNITE DE DECISION

5. Feedback

Le feedback positif (FB +) a pour rôle de renforcer les bonnes réponses, le feedback négatif (FB -) d'indiquer les erreurs à l'apprenant de telle sorte qu'il se corrige.

FB +

[5] [11] - Est-il approprié à l'âge, au niveau et à la situation?

[11] - N'y a-t-il pas de renforcement positif de mauvaises réponses ?
Exemple : apparition d'un graphique plus beau et plus complexe en cas d'erreur.

[7] - Suit-il directement toutes les bonnes réponses de l'apprenant ?

Cependant, dans le cas de réponses trop fréquentes, on conseille de donner au début des FB + pour chaque réponse, puis de les espacer.

FB -

[7] [11] - L'évaluation de la réponse est-elle neutre ?
(ou provoque-t-elle de l'anxiété, de la peur ou du rejet par son caractère agressif, moqueur, etc...)

[7] [8] - Le feedback est-il le plus souvent
. stéréotypé ? Faux
. spécifique ? Dire pourquoi ou en quoi ...

Exemple 1 : R attendue "50 km/h"
R de l'élève "100 km"
FB spécifique : "tu te trompes d'unité."

Exemple 2 : R.A. "6 x 4 = ?" "24"
R.E. "10"
FB spécifique : "Non, tu additionnes."

. renvoyant à l'analyse par l'élève?

		-	0	+
[10] [2] [3]	- Après avoir dit que la réponse était fausse, le programme propose-t-il - directement la B.R. ? - de répondre à nouveau à la question ? - un indice supplémentaire, puis de ... ? - de relire l'information donnée avant la question, puis de ... ? - une nouvelle manière de présenter l'information, puis de ... ? - une nouvelle information appropriée au type d'erreur commise, puis de ... ?			

6. Branchement

Quels types d'information le didacticiel peut-il traiter ? Comment (quels traitements est-il capable de réaliser sur ces informations) ? Pour quels types d'actions ? L'évaluation porte ici sur la souplesse d'adaptabilité de l'unité de décision du didacticiel.

Le modèle de la page suivante propose une évaluation de la qualité d'un didacticiel reposant sur les décisions qu'il est capable de prendre à partir de traitements qu'il effectue.

On estime que, plus un didacticiel est capable de traiter un grand nombre d'informations pour aboutir à un grand nombre de décisions individualisées, plus il utilise au mieux les possibilités de l'outil "ordinateur", il crée ainsi par rapport au livre, au tableau, à la T.V., sa spécificité (l'interactivité) et justifie son utilisation à l'école.

Le modèle est composé de trois parties principales :

a. INFORMATIONS

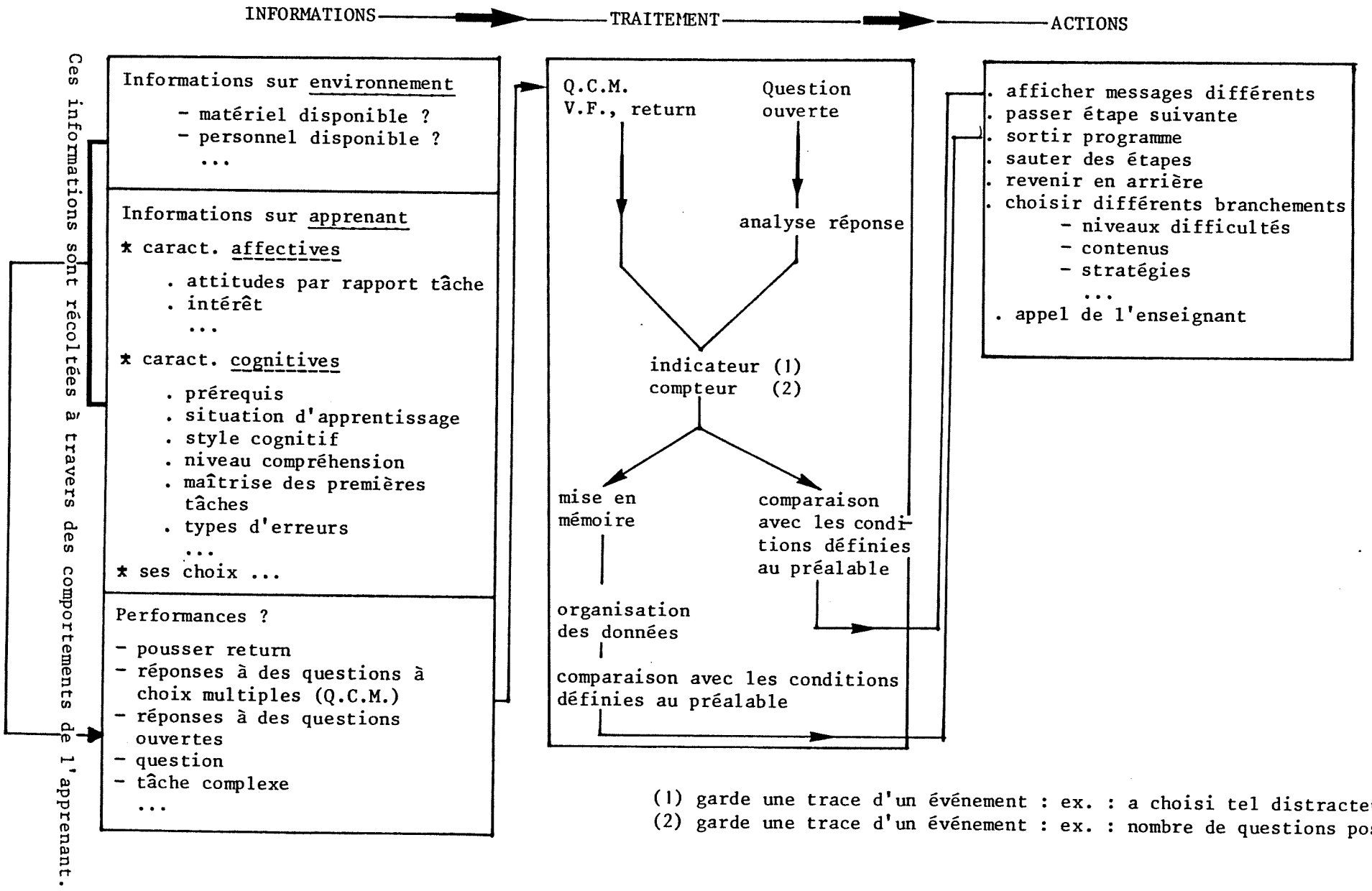
Cette partie propose un échantillon d'informations qu'un didacticiel peut prendre en charge, soit à l'entrée du programme, soit en cours d'exécution.

b. TRAITEMENT

Le programme traite ensuite les données recueillies : il repère les réponses attendues, les enregistre et les organise.

c. ACTIONS

Enfin, la troisième partie passe en revue différentes actions que le programme pourrait exécuter en fonction des informations dont il dispose sur l'apprenant et sur son environnement.



(1) garde une trace d'un événement : ex. : a choisi tel distracteur
(2) garde une trace d'un événement : ex. : nombre de questions posées

Tutoriel ramifié : cfr. ci-dessus. Les systèmes experts (I.A.) permettent de générer des branchements nouveaux qui n'avaient pas été "programmés" auparavant.

Modélisation : l'apprenant est confronté à des données qu'il doit tenter de comprendre en établissant un modèle et en le testant.

Simulation : l'ordinateur sert de simulateur : il place l'apprenant devant une situation, une tâche ou un environnement "réels". Celui-ci est alors invité à faire varier des paramètres.

Programmation : l'apprenant prend le pouvoir sur la machine. L'ordinateur est alors programmé par l'étudiant ; cette activité permet l'accentuation des habiletés d'analyse et de résolution de problèmes.

2. LES OBJECTIFS

		-	0	+
	2.1. Les objectifs sont-ils explicités ? Si oui, lesquels ? généraux ... intermédiaires ... opérationnels ... (précis et en terme de comportements observables d'élèves)			
	2.2. Ces objectifs s'inscrivent-ils dans un programme d'étude ? Si oui, lequel ? ...			
13	2.3. Le type de didacticiel (cfr. point 1) convient-il à l'objectif ?			
13	2.4. La démarche suivie est-elle cohérente par rapport aux objectifs ?			
13	2.5. La matière et le domaine sont-ils adéquats par rapport aux objectifs ?			
	2.6. Les résultats (acquis des élèves) correspondent-ils aux objectifs explicités ?			
	2.7. Ces objectifs correspondent-ils à ceux que l'enseignant poursuit à propos de cette matière ? Si non, celui-ci les juge-t-il pertinents ?			
9	2.8. Les objectifs sont-ils présentés à l'apprenant dans un langage compréhensible par lui ?			

3. INSERTION DU DIDACTICIEL

- Le didacticiel entraîne-t-il des activités annexes ? (Travail en labo, etc...) Si oui, lesquelles ?
- Le didacticiel peut-il offrir des prolongements en classe (discussion en groupe, etc...) Si oui, lesquels ?

	-	0	+

- Quelle place prend-il par rapport au professeur ?
  Dans quelle(s) fonction(s) l'aide-t-il ?
  (La réponse à cette question permet de cerner les fonctions que l'enseignant doit lui-même prendre en charge pour compléter le didacticiel).

  . éveiller l'intérêt ? (ex. : simulation)
  . présenter de l'information ? (cours)
  . consolidation d'acquis ? (ex. : répétitifs)
  . contrôle des connaissances ? (test)
  . diagnostic ?

- Vient-il en plus du curriculum normal ?
  Ou l'unité matière fait-elle partie du curriculum ?

4. DOCUMENTATION

- Existe-t-il un livre du maître ?
  Si oui, a-t-on des renseignements sur

  . la façon dont le programme fonctionne ?
  . la manière d'utiliser le programme ?
    - en tant que tel (le présenter, etc...) ?
    - la stratégie pour l'insérer dans l'activité de la classe ?
    - son insertion dans le cours (curriculum) ?
  . la façon de modifier le programme ?
  . les objectifs, les prérequis, le temps de passation moyen, etc ...

[N.B. Pour évaluer le livre du maître, utilisez le test suivant : uniquement à partir de la documentation, pouvez-vous dire avec précision de quel type de didacticiel il s'agit ? (cfr. point 1)

- Existe-t-il un manuel pédagogique de l'étudiant ?
  Si oui, comprend-il les explications suivantes :

  . comment faire tourner le didacticiel ? (indiquer les touches utiles, etc...)
  . quels sont les objectifs poursuivis ?
  . quelles sont les procédures à utiliser pour sortir du programme, accéder à une information antérieure, etc... ?

5. CONTROLE DU PROGRAMME

Pour l'étudiant

[3] - L'étudiant peut-il arrêter le déroulement du didacticiel et en sortir quand il veut ? (STOP)

[9] - L'étudiant peut-il faire appel au besoin à une fonction d'aide ? (AIDE)

[3] [1] - Peut-il revenir à la description des instructions de base à tout moment ?

		-	0	+
[3]	- Peut-il s'apesentir sur un chapitre plus difficile ?			
[3]	- Peut-il sauter un chapitre ?			
[8]	- Peut-il, à chaque moment, savoir où il en est et ce qui reste à faire ?			
[3]	- Peut-il revenir directement là où il a quitté le cours lors d'une séance précédente ?			
[3]	- Peut-il contrôler la vitesse de présentation ?			
[3]	- Peut-il contrôler la quantité d'information ?			
[3]	- Peut-il contrôler la quantité de problèmes posés ?			
[3]	- Peut-il contrôler la quantité d'exemples proposés ?			
[3]	- Peut-il choisir un exercice parmi un ensemble proposé ?			
[3]	- Peut-il choisir une expérience parmi un ensemble proposé ?			
[3]	- Peut-il choisir les données sur lesquelles il va travailler ?			
[3] [2]	- Peut-il facilement effacer ou corriger ses erreurs tant qu'il n'est pas sûr de sa réponse ?			
[3]	- Peut-il choisir, à l'aide d'un menu, certains segments du programme ?			
	Pour le professeur			
	- Peut-il modifier le programme ?			
	- Peut-il supprimer des passages ?			
	- Peut-il modifier le contenu ?			
	- Peut-il modifier la stratégie ?			
	- Peut-il modifier les exemples illustratifs ?			
	6. EVALUATION DE L'EFFICACITE DU DIDACTICIEL			
	- Existe-t-il des données expérimentales qui montrent que le didacticiel atteint bien ses objectifs ?			
	- Existe-t-il des données expérimentales qui montrent que le didacticiel atteint des objectifs différents avec des groupes différents ou dans des contextes d'utilisation différents ?			

	-	0	+

7. EVALUATION DES APPRENTISSAGES

- Existe-t-il une stratégie d'évaluation des apprentissages des élèves ? (Ex. : "prétest-postest")
- L'évaluation prévue est-elle valide ?
  fidèle ?
- Est-elle annoncée à l'apprenant ?

8. TRACES POUR L'ENSEIGNANT

- Peut-il garder une trace des résultats de l'élève ?
- Peut-il garder une trace du cheminement de l'élève ?
- Peut-il garder une trace de l'endroit où il est arrivé ? (Pour y revenir à la leçon suivante)

[8] - Peut-il disposer d'un diagnostic ?

[8] - Peut-il disposer d'un rapport sur les progrès de l'élève ?

- Peut-il disposer d'une proposition d'activité de prolongement ?

9. DIVERS

- Le programme génère-t-il au hasard les items ?
  exercices ?
  mots ?
  chiffres ?

  Cet aspect est capital pour les exercices répétitifs si l'on veut qu'un même élève les utilise plusieurs fois.

Annexe 1. Comparaison de lisibilité de diagrammes

L'exemple ci-dessous donné par RICHARDEAU (17) concerne l'illustration de l'évolution de l'agriculture en France depuis 1940.

Le tableau des chiffres est difficilement interprétable directement -1- et l'on cherche à l'illustrer.

"Voici -de 2 à 8- les formules "originales" et quelquefois "recommandées" qu'on peut rencontrer dans certains manuels.

Quelles sont les questions pertinentes à cette information ? L'information fait état de l' "ordre" du temps et de "différentes" cultures. Les questions pertinentes sont donc :
1. "Quelles sont les cultures qui ont augmenté, quelles sont celles qui ont diminué ?"
2. "Comment les cultures se classent-elles suivant leur évolution ? En combien de temps pouvez-vous répondre sûrement et complètement ? Quel dessin regardez-vous ? Ne regardez-vous pas le tableau de chiffres ?"

((17) p.92)

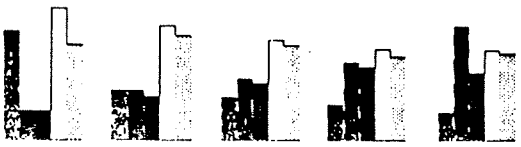
1840	1880	1910	1938	1960	
28	14	11	8	6	Jachères
7	14	16	21	29	Fourrages
7	12	15	20	18	Plantes sarclées
37	32	30	27	24	Céréales
21	28	28	24	23	Blé

Surface cultivée (%) en France

1

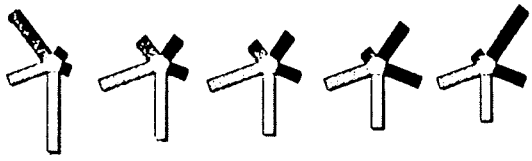
2

3

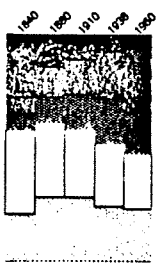


4

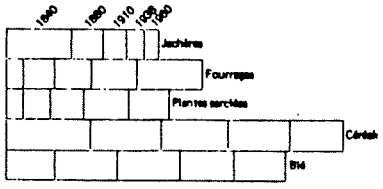
5



6



7



8

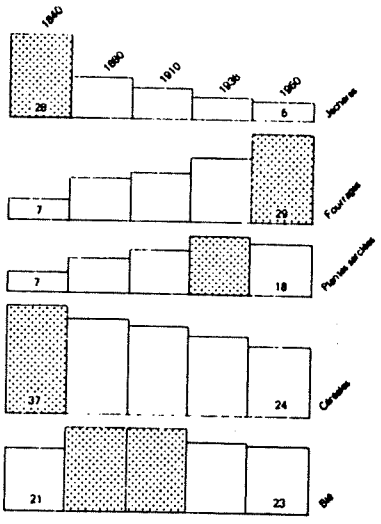
" Il n'y a qu'une seule construction qui réponde instantanément à toutes les questions. C'est la construction "normale" x, y, z. -11-

1. Tout problème graphique est un problème de relation. Il peut donc être construit sous la forme d'un tableau à double entrée xy. Les cases ne doivent contenir que des réponses "oui-non", des nombres ordinaux ou cardinaux. Ces réponses sont le z de l'information.

2. Le but d'une construction graphique est la découverte des groupements ou des ordres, en x et en y, que les données z construisent.

3. La seule construction qui, dans tous les cas, permet cette découverte est la construction xyz. C'est aussi la plus simple. Il suffit :
   a) de reproduire la disposition du tableau, c'est-à-dire de placer en x et en y sur le graphique les entrées y et y du tableau ;
   b) de transcrire en z les nombres par une variation de taille ou de valeur ;
   c) de permuter les lignes et/ou les colonnes pour découvrir les groupements ou les ordres recherchés.

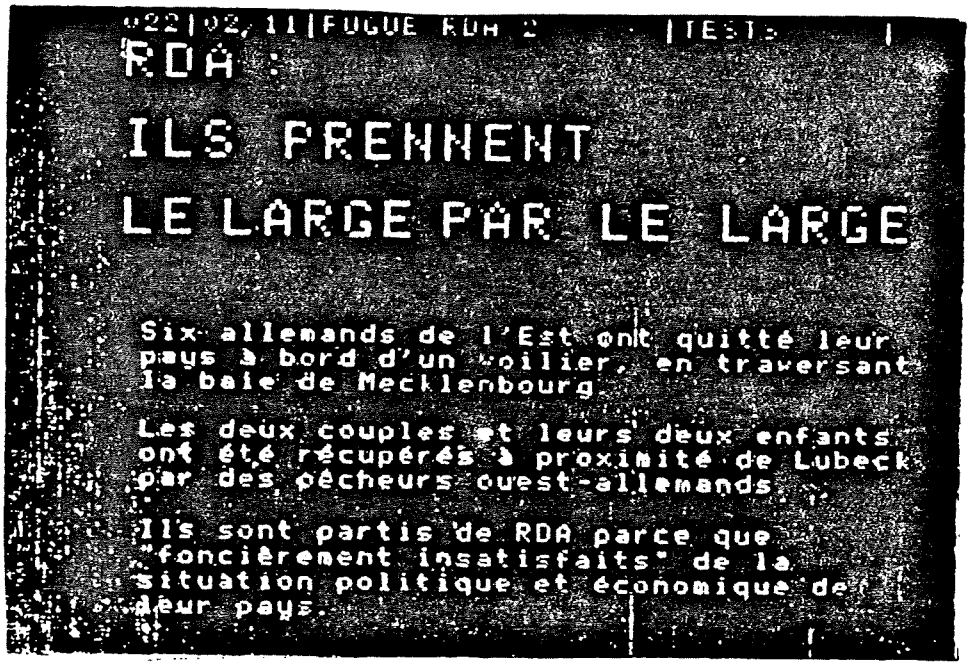
La graphique moderne présente deux originalités par rapport aux habitudes acquises de la graphique classique :
- elle est infiniment plus simple et devient à la portée de tous
- elle construit une image mobile et devient par là un instrument de réflexion dont les propriétés se révèlent exceptionnelles, aussi bien dans la recherche scientifique que dans l'animation des classes et la motivation des écoliers, depuis le niveau le plus élémentaire jusqu'aux étudiants de facultés." (17) p. 93.



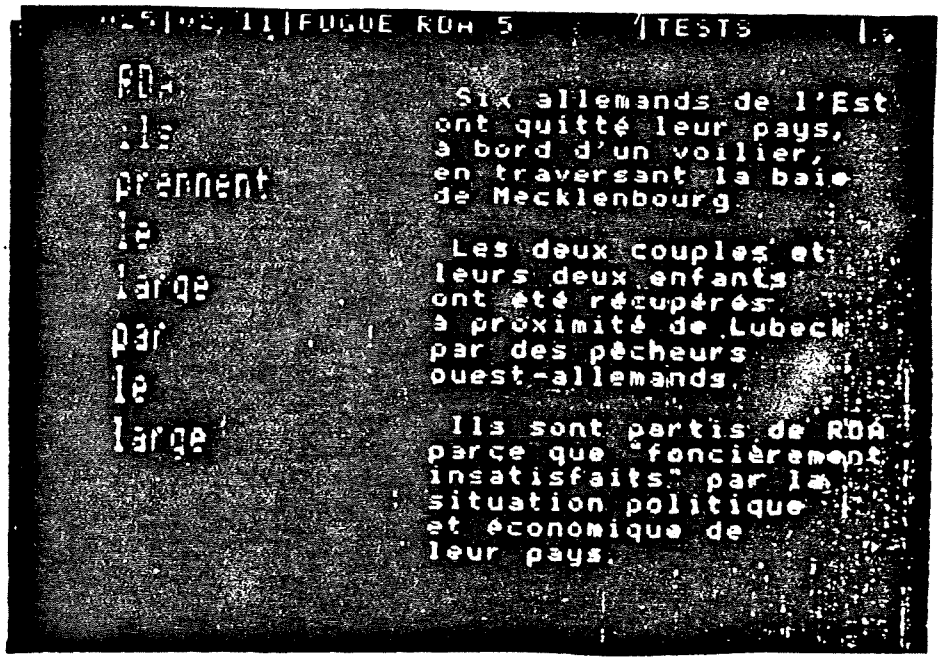
11

Annexe 2. Disposition du titre et du texte sur l'écran (6)

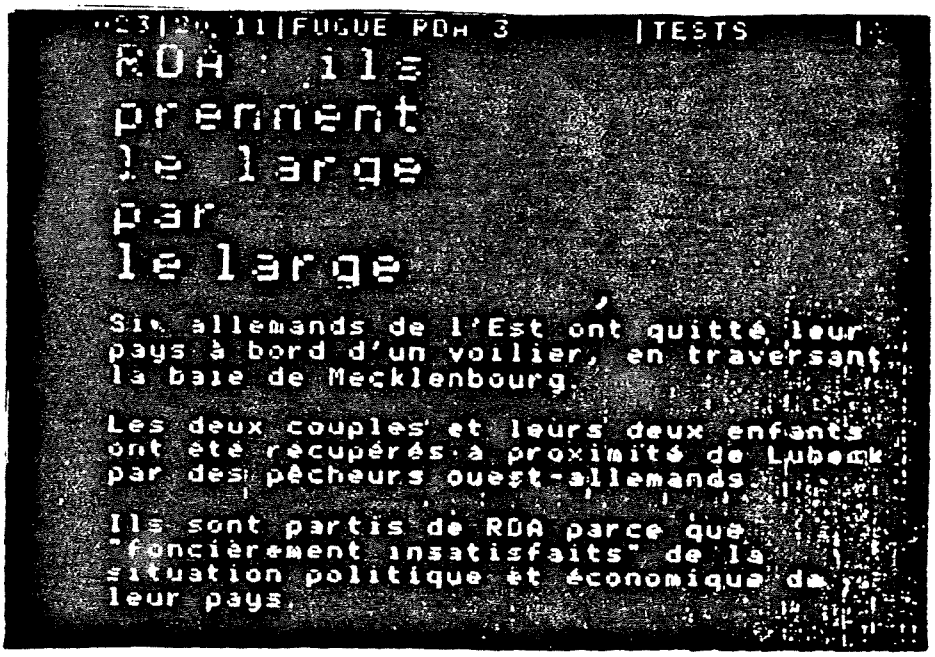
Voici trois types d'organisation de la page écran testées par le groupe CFPJ. Pour chaque écran, nous disposons de son pourcentage d'acceptation et de sa place (sur 10 écrans testés).



Ecran 2 = 65% de oui - 1er sur 10



Ecran 5 = 20% de oui - 9° sur 10



Ecran 3 = 14% de oui – 10° sur 10

Bibliographie

(1) ALBERTINI, J-M. ; Vers une informatique de formation, Education Permanente, n° 70-71, déc. 1983, p. 51-72.

(2) BESTOUGEFF, H. ; FARGETTE, J-P. ; Enseignement et ordinateur, F. Nathan, Paris, 1982.

(3) BLOOM, B.S., Caractéristiques individuelles et apprentissages scolaires, Bruxelles, Labor, 1979.

(4) BURKHARDT, H. ; FRASER, R. ; WELLS, C. ; Teaching Style and Program Design in Computer and Education, vol. 6, 1982, pp. 77-84.

(5) COHEN, V.B. ; Criteria for the Evaluation of Microcomputer Courseware, Educational Technology, janvier 1983, pp. 9-14.

(6) DE LANDSHEERE, G. ; Education et ordinateur, Bulletin de l'U.L.B. et de l'U.A.E., n° 30, septembre 1982.

(7) DUBUC, L. ; Classification des A.P.O., Ministère Education Québec, novembre 1982.

(8) DURETT, J. ; TREZONA, J. ; The Elements of Color Vision and Their Implications for Programmers, Pipeline, vol. 8, n° 1983, pp. 33-35.

(9) Groupe de Recherche CFPJ, Lisibilité et écriture télématique, Communication et Langage, 56, 1983, pp. 64-83.

(10) HADJI, Ch. ; Neurobiologie et pédagogie, Revue Française de Pédagogie, n° 67, 1984, p. 38.

(11) JAY, T.B. ; The cognitive Approach to Computer Courseware Design and Evaluation, Educational Technology, janvier 1983, pp. 22-26.

(12) KOSEL, M. ; JOSTAD, K. ; Designing the Display, Pipeline, Vol. 8, n° 2, 1983, pp. 29-32.

(13) LATHROP, A. ; GOODSON, B. ; Courseware on the Classroom, Addison-Wesley, 1983.

(14) MATAIGNE, B. ; Guide d'évaluation de didacticiels, Ministère Education Québec, mars 1983.

(15) MATAIGNE, B. ; L'évaluation des didacticiels : pourquoi ? comment ?, conférence L21 au congrès "l'Ordinateur et l'Education", Montréal, 1983.

(16) PREECE, J. ; A Study of Pupils Graph Concepts with a Qualitative Interactive Graph Sketching Program, Computer and Education, vol. 8, n° 1, 1984, pp. 159-163.

(17) RICHARDEAU, F. ; Conception et production des manuels scolaires, Paris, Retz, 1981.

(18) SADLER, C. ; EISENBACH, S. ; Selecting Microcomputers for Schools, Comput. Educ., vol. 8, n° 1, 1984, pp. 165-171.

(19) SAINTE-MARIE, J. ; Microscope 5, Legatoscope, Guérin, Montréal, 1983.

(20) STEINBERG, E.R. ; Revieuwing the Instructional Effectiveness of Computer Courseware, Educational Technology, janvier 1983, pp. 17-19.

(21) TRUBERT, I. ; Produire un bon didacticiel, Journal de la Formation Continue et de l'E.A.O., n° 171, 1984, pp. 14-15.

(22) Guide d'applications pédagogiques, T.V. Ontario, Gouvernement Québec, 1983.

16

CHAPITRE 4

TOURS DE MAIN DE PROGRAMMATION

AVERTISSEMENT

Nous savons à présent que "programmer" ce sera "réd
à suivre (comportant instructions d'actions et structures d
un exécutant (gestionnaire de casiers, manipulateur d'
communiquant avec l'extérieur).

Sachant cela, je vais maintenant présenter un certain nombre d'exemples illustrant la démarche de traitement informatique d'une tâche. Le chemin (nous le savons) sera à chaque fois balisé par les étapes mises en évidence : QUOI FAIRE ?, COMMENT FAIRE ?, COMMENT FAIRE FAIRE ? et COMMENT DIRE ?.

Cette dernière étape, traduction en Pascal de la marche à suivre obtenue sous forme GNS, ne sera cependant décrite qu'au chapitre suivant.

Les premières tâches abordées seront particulièrement simples (pour ne pas dire ridicules). Elles seront peu motivantes, conduisant en général à des programmes qui ne seront pas utilisés. C'est qu'il s'agit à présent (connaissant les outils de conception d'une marche à suivre et les instructions d'actions élémentaires permises pour l'exécutant-ordinateur), non de traiter des tâches utiles ou passionnantes, mais d'acquérir un certain nombre de tours de main de la programmation.

Il faut donc "jouer le jeu" : lorsque l'apprenti-plafonneur débute dans le métier, on ne lui confie pas d'emblée le traitement d'un bâtiment dans son ensemble; on lui fait acquérir les "trucs du métier". Il travaille "pour rire" (même si ce n'est pas très amusant de passer un long moment à travailler "pour rien"). C'est pareil pour l'apprenti-programmeur: il ne s'agit pas (encore) de programmer des tâches intéressantes, utiles ou motivantes ! Il faut (bêtement) acquérir, à l'occasion de tâches débiles, les premiers tours de main de la programmation.

A côté des exercices proposés (et résolus), on trouvera également une série d'exercices qui permettront à l'apprenti-programmeur de se faire la main. C'est uniquement par un travail personnel de résolution de ces exercices que vous pourrez progresser. La simple "compréhension" des solutions apportées aux exercices traités ici est illusoire : c'est seulement en agissant que l'apprentissage sera effectif.

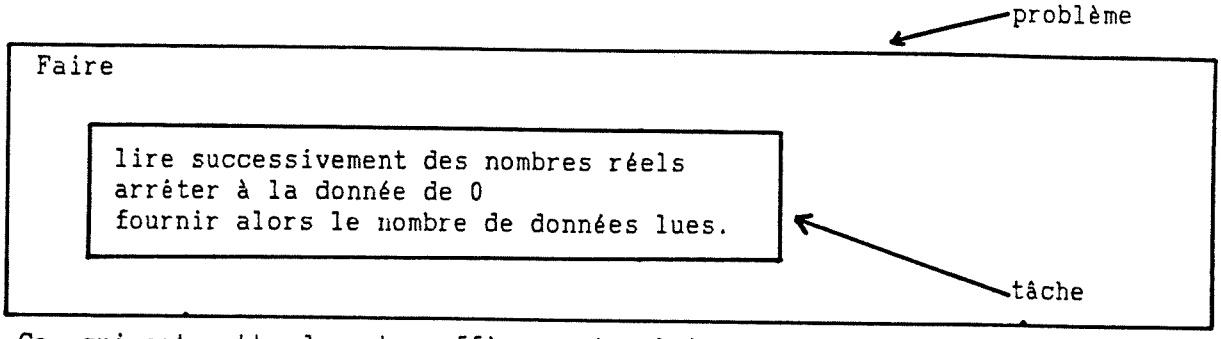
TRAITEMENT DES EXEMPLES

Problème 1

La première étape (on le sait) consistera à préciser quelle est la tâche proposée : que faut-il faire (ou plutôt faire faire). Les tâches abordées ici sont tellement simples, que cette étape de spécification sera immédiate. Il ne faudrait pas en déduire qu'il en sera toujours ainsi : dans le cas des vrais problèmes (de taille réelle), c'est en général l'étape la plus longue et la plus difficile.

3054

Enoncé (Quoi faire ?)



Ce qui est attendu est suffisamment clair pour ne pas nécessiter de développements plus longs : on peut imaginer que quelqu'un ("l'utilisateur") va nous dicter des nombres; lorsque c'est 0 qui sera fourni, on attend simplement que nous répondions en précisant combien de nombres nous auront été fournis au total. Cette tâche est débile; elle ne devient un problème que parce que son énoncé commence par "faire" ...Le problème, c'est de rédiger la marche à suivre qui expliquera à l'exécutant ordinateur comment accepter des nombres (fournis à travers la porte-clavier par l'utilisateur), les compter et donner (sur la fenêtre-écran) le résultat à l'apparition de 0.

Il reste dès lors à passer à l'étape suivante : comment réagirions nous, face à cette tâche ? Quelles seraient nos stratégies ?

Comment faire ?

La recherche de stratégies pour venir à bout de ce travail ne nous conduit pas bien loin pour l'instant : la tâche est vraiment trop élémentaire. En réalité, la seule chose à préciser et à percevoir dans notre comportement c'est : quelles informations retiendrions-nous (que garderions nous "en tête") si l'on nous demandait d'effectuer la tâche décrite.

> Cette réflexion débouchera dans l'étape suivante sur l'établissement de la liste des casiers (variables) nécessaires à l'exécutant-ordinateur. Nous savons en effet qu'il est amnésique et qu'à tout instant, sa seule "mémoire", c'est le contenu des casiers que nous aurons choisi d'utiliser au départ.

Ainsi, si l'on me dicte des nombres avec comme consigne préalable de dire, à l'apparition du nombre 0, combien j'en ai reçu au total il est inutile que j'essaye de retenir tous ces nombres. Simplement je retiendrai chacun d'eux un court instant, le temps de le compter et de m'apercevoir que c'est ou ce n'est pas 0.

Au prise avec l'exécution de cette tâche, nous garderions donc en mémoire

- chaque donnée pendant un court instant
- le décompte des données qui évoluera au fur et à mesure qu'elles sont fournies (nous les comptons!).

On pourrait ajouter à cette liste le fait suivant : c'est le nombre 0 qui nous fera arrêter les lectures; ceci reste également présent dans notre mémoire. Cette connaissance n'est cependant pas susceptible d'être modifiée en cours de travail, et cela à la différence des diverses données et de

leur décompte. Une telle quantité constante ne figurera pas dans la liste de ce que nous retiendrons, liste qui donnera naissance aux "casiers" (à contenu variable) indispensables pour l'exécutant-ordinateur.

> Cette étape de recherche de stratégies, où nous devons être capables, face à une tâche, d'expliciter complètement comment nous nous y prenons sera en général moins anodine. Pensez à la difficulté d'expliquer complètement et de manière méthodique la manière dont vous triez des nombres, dont vous écrivez en toutes lettres un nombre, ou dont ... vous comptez les points au tennis!
>
> Pour l'instant, la seule utilité de cette étape, c'est se regardant faire aux prises avec les tâches demandées, d'être capable de préciser les informations que nous gardons à l'esprit pour en venir à bout. Nous saurons ainsi, à l'étape suivante quels sont les casiers nécessaires.

Nous pouvons à présent passer à la suite : nous allons tenir vraiment compte des caractéristiques de l'exécutant, pour rédiger la marche à suivre qui lui expliquera comment traiter cette tâche à notre place.

Comment faire faire ?

L'exécutant est un gestionnaire de casiers. Il est impératif, dès lors de commencer par préciser quels casiers lui seront indispensables pour effectuer la tâche désirée. Rappelons que c'est un amnésique, que nous ne pourrons lui parler qu'à l'impératif présent et que sa seule "mémoire", à tout instant, c'est ... le contenu de ces casiers !

Liste des variables (casiers)

Un casier sera évidemment nécessaire pour accueillir les données à lire. Comme chacune de ces données peut être oubliée à l'apparition de la suivante, cet unique casier suffira : la donnée lue y résidera le temps nécessaire avant d'être remplacée par la suivante. Comme ces données seront des nombres réels, le casier destiné à les recevoir l'une après l'autre sera de type réel. Pour rappeler ce qui y sera contenu, je l'étiqueterai NombreFourni.

Ainsi, les caractéristiques de ce casier sont les suivantes :

étiquette (nom)	: NombreFourni
type	: réel
contenu	: chaque donnée lue.

De plus, il me faut faire retenir le décompte des données lues. Pour cela je prévoirai un casier qui s'augmentera de 1 à chaque nouvelle donnée; il sera de type entier et je le baptise Compteur, puisqu'il servira à compter!

Ses caractéristiques sont donc les suivantes :

étiquette	: Compteur
type	: entier
contenu	: le nombre de données.

> A ce stade, nous gardons toute liberté quant au choix du nom des casiers utilisés : c'est seulement lors du codage de la marche à suivre en Pascal que les contraintes syntaxiques feront leur apparition. Nous sacrifierons cependant, dès à présent, aux

prescriptions grammaticales de Pascal concernant les noms des variables : ils ne comporteront que des lettres (majuscules ou minuscules) et des chiffres, à l'exclusion de tout autre symbole (comme l'espace, le tiret, ...) et débuteront par une lettre. C'est pour cette raison que je parle ci-dessus de NombreFourni et pas de Nombre Fourni.
Au delà de ces (petites) contraintes orthographiques, le plus important, quant au choix des noms de variable, c'est bien sûr d'opter pour des termes liés aux informations qui y seront contenues. Mieux vaut donc NombreFourni que X, N ou Salaire et Compteur que U, I ou Age !

On le voit, ces casiers correspondent bien aux "choses" que je devais à tout moment garder en tête pour effectuer le travail:

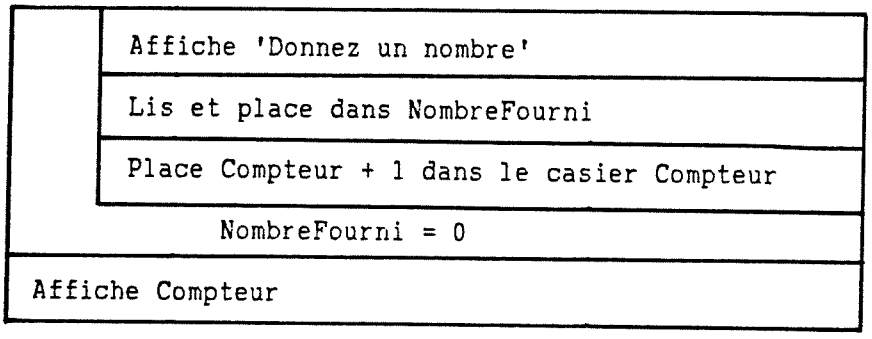
je retenais	casiers
chaque donnée (un instant)	NombreFourni
le nombre de données lues	Compteur

Il reste seulement à préciser comment ces casiers seront utilisés: en un mot quelle est la

Marche à suivre

Nous allons demander à l'exécutant de répéter essentiellement les mêmes actions : d'afficher à la fenêtre-écran un court "message" (1) invitant l'utilisateur à donner un nombre, d'aller lire ce nombre à partir de la "porte-clavier" (pour le déposer dans le casier NombreFourni) et de le compter (en augmentant le contenu du casier Compteur). Et cela, il devra le répéter jusqu'au moment où le nombre fourni (et déposé dans NombreFourni) sera 0. Il suffira alors de lui demander d'afficher sur la vitrine-écran, à l'intention de l'utilisateur, le contenu du casier Compteur (qui contiendra alors le nombre de données lues).

Ainsi la marche à suivre pourrait être (sous forme GNS) :



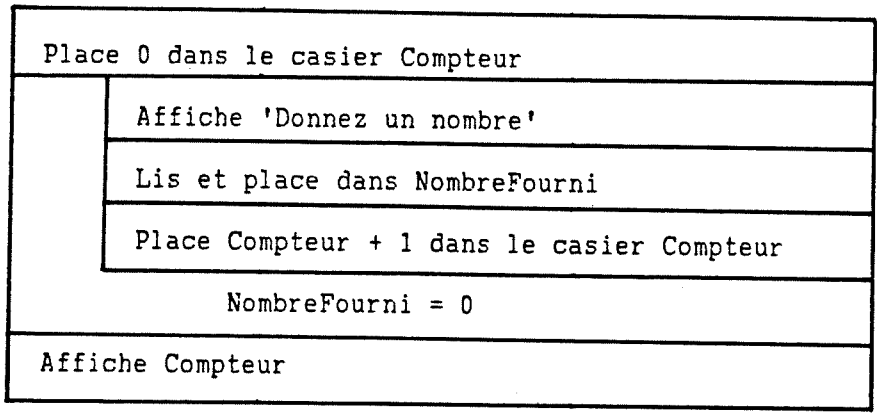
Il reste cependant un petit problème : lorsque l'exécutant commence l'exécution de cette marche à suivre, les casiers contiennent n'importe quoi (n'importe quelle information du type correspondant au type du casier envisagé). Il est facile de voir, dans ces conditions que si le casier Compteur contient (primitivement et par hasard) l'entier 12 et si

(1) *du point de vue de l'exécutant, il s'agit bien sûr d'une information constante de type chaîne de caractères.*

l'utilisateur fournit les nombres 3.5, -4.66 et 0, il lui sera répondu qu'il a fourni 15 nombres au total (puisqu'on commence à compter avec un compteur qui contient déjà 12 !).

Il est donc impératif que l'exécutant commence bien son travail avec un Compteur contenant 0 et que, dès lors, je lui demande, avant de commencer à répéter les lectures et comptages successifs, d'y placer 0. Cette étape de réflexion à propos du contenu préalable des casiers est indispensable : c'est l'initialisation des casiers.

Ici, le contenu primitif de Compteur doit être 0; celui de NombreFourni est sans importance, puisque la première chose que je commande de faire à propos de ce casier c'est d'y placer une information (lue au clavier). Dès lors la marche à suivre complète et conforme aux spécifications est



Remarques

Les outils (de la table de travail) mis en oeuvre dans ce problème sont

+ qui fournit la somme de deux nombres
= qui permet la comparaison de deux informations

Disposant à présent de la liste des casiers nécessaires et de la marche à suivre correspondante, il resterait à traduire tout ceci dans le langage compromis "compréhensible" par l'ordinateur. Nous réserverons au chapitre suivant la réalisation de cette étape.

On le verra, l'essentiel est à présent terminé : nous disposons, sous forme GNS, de la marche à suivre qui permettra à l'exécutant d'effectuer la tâche décrite à notre place. Et il ne nous reste plus qu'à ... passer à la tâche suivante.

----- 0 -----

Problème 2

Enoncé (Quoi faire)

Faire

lire successivement des mots
arrêter à la lecture de STOP à condition que ATTENTION ait été fourni (n'importe quand) auparavant
donner alors le nombre de mots lus au total.

On le voit à nouveau, cette tâche est particulièrement simple et peu intéressante. Le problème de programmation auquel elle conduira sera ... moins simple et ... (je l'espère) plus intéressant.

Comment faire ?

Comme pour la tâche précédente, ce qui importe c'est, se regardant faire aux prises avec ce travail, de déterminer quelles sont les informations que nous sommes forcés de retenir et qui sont susceptibles de se modifier.

Deux approches (au moins) sont sans doute possibles :

1) On ne retient qu'un court instant le mot lu. Dans un premier stade, on est seulement attentif à l'apparition de ATTENTION; dès réception de ce mot, on passe alors à un second stade où l'on est attentif à l'arrivée de STOP qui arrête tout le processus. On compte les mots au cours de la première étape et on continue ce décompte au cours de la seconde.

ou encore

2) On peut aussi retenir un instant chaque mot mais "retenir" à tout moment si ATTENTION a ou n'a pas encore été fourni. On s'arrête lorsque STOP apparaît, à condition qu'on se souvienne que ATTENTION est passé.
De toute manière, on compte les mots à chaque fois.

Quelle que soit l'approche envisagée, on retient évidemment les mots ATTENTION et STOP (auxquels on compare les mots lus) mais ces deux informations ne se modifieront pas plus au cours de la tâche que le 0 du problème précédent.

Comment faire faire ?

Première solution

Comme précédemment, il me faut d'abord, sur base des informations que nous étions forcés de retenir à l'étape du "Comment faire?", décider quels seront les casiers indispensables.
Il en faudra seulement deux

- celui qui contiendra successivement chacun des mots lus à partir du clavier, que je baptiserai pour cette raison MotLu. Il doit être du type chaîne de caractères.

A nouveau, au delà des contraintes orthographiques qui seront imposées lors de la traduction en Pascal et que j'ai décidé de respecter dès à présent, l'important, c'est de choisir un nom de variable qui informe sur son contenu : j'aurais pu choisir MotFourni, Mot, Donnée, ...

- celui qui servira à compter : Compteur, de type entier.

Rappelons bien, que tout au long du traitement de ce problème, chaque fois que nous parlerons de "mot", il s'agira bien entendu, du point de vue de l'exécutant, d'une chaîne de caractères.
De plus, lors de la définition d'un casier de ce type, nous préciserons la longueur maximale des chaînes qui pourront y prendre place. Il s'agit là d'une contrainte qui n'apparaîtra vraiment que lors de la traduction en Pascal. Nous en tenons compte dès à présent pour faciliter ce travail de traduction (Comment dire ?). On verra dans la suite que cette longueur maximale à préciser est comprise entre 1 et 255.

Ainsi donc je peux à présent dresser la

Liste des variables

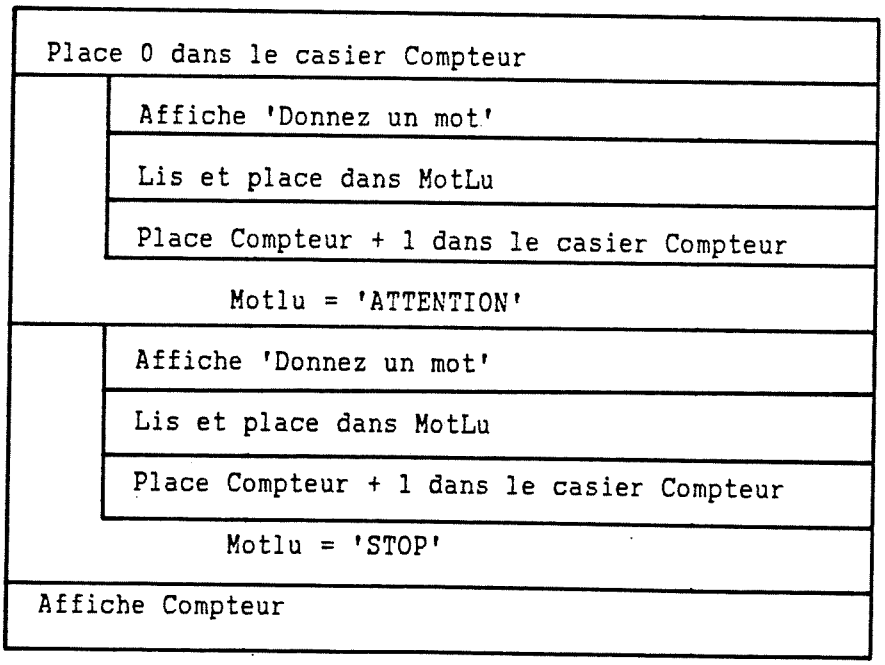
MotLu, de type chaine d'au plus 20 caractères
Compteur, de type entier

Il reste à indiquer comment ces deux casiers seront utilisés en fournissant la

Marche à suivre

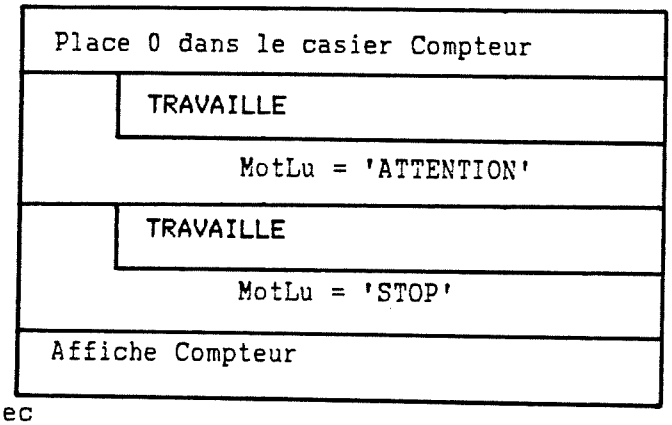
Nous allons commander deux répétitions successives des mêmes actions: lire une donnée (pour la déposer dans MotLu) et la compter (grâce au Compteur). La première répétition s'interrompra lorsque le mot lu sera 'ATTENTION', faisant alors place à la répétition suivante qui s'arrêtera à 'STOP'. Il ne restera plus alors qu'à demander l'affichage du contenu du Compteur... qu'il ne faudra pas oublier, dès le début, d'initialiser.

Ceci conduit au GNS suivant :



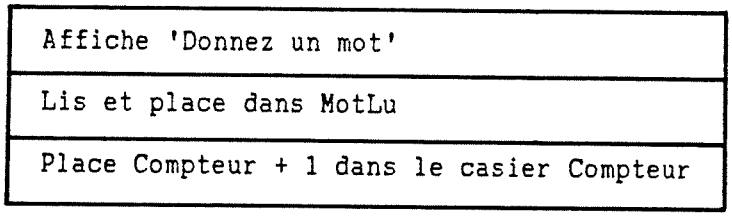
On constate que les deux structures répétitives demandent la répétition du même groupe de trois actions. Nous pourrions donc, pour sacrifier à une certaine paresse, désigner par une seule instruction ce groupe de trois instructions élémentaires. Notre marche à suivre comporterait alors une instruction d'action complexe (appel de procédure) explicitée dans une marche à suivre annexe (procédure).

Ainsi, en désignant par "Travaille" l'instruction complexe (qui se scindera en trois instructions élémentaires), le GNS devient :



avec

Procédure annexe : TRAVAILLE



Il ne faudrait pas croire que l'utilisation des procédures ne se fera que lorsqu'une portion de marche à suivre apparaîtra à divers endroits, pour sacrifier (comme ci-dessus) à une certaine paresse. On le verra, ce concept permettra d'incarner dans nos analyse la démarche descendante.

Rappelons que face à cette marche à suivre, l'exécutant s'intéressera d'abord au texte principal, en détournant son regard vers les explications fournies dans la procédure annexe chaque fois qu'il rencontrera l'instruction inconnue TRAVAILLE.
Après avoir exécuté les actions prescrites dans l'annexe explicative, il reviendra à la suite de la marche à suivre principale.

Deuxième solution

Dans cette seconde approche, une seule répétition sera commandée. Elle s'arrêtera lorsque le mot lu sera 'STOP' et que l'exécutant "gardera le souvenir" du passage de 'ATTENTION'. Le seul problème, c'est donc qu'il "garde en mémoire" le fait que 'ATTENTION' a ou n'a pas encore été lu. Mais, pour cela, il suffit d'un casier dont le contenu signalera que ce mot a (ou n'a pas) été fourni. Ce casier, je le baptiserai Signal; j'ai le choix en ce qui concerne son type : l'important, c'est qu'il contienne une certaine information tant que 'ATTENTION' n'a pas été reçu et qu'à la

réception de ce mot son contenu se modifie. Ainsi, il pourrait être de type entier, contenant primitivement 0 (ou toute autre valeur entière) et basculant à 1 (ou à toute autre valeur différente de celle s'y trouvant) lors du passage de 'ATTENTION'. Il pourrait aussi être de type réel, mais j'ai choisi ici le type chaîne d'au plus 5 caractères : Signal contiendra la chaîne 'VERT' tant que le mot 'ATTENTION' n'aura pas été reçu et ce contenu deviendra 'ROUGE' (pour le rester) à la réception du mot 'ATTENTION'.

Ainsi, on vérifiera à la lecture de chaque mot qu'il s'agit oui ou non de 'ATTENTION' pour faire éventuellement basculer le contenu de Signal de 'VERT' à 'ROUGE'. Les lectures (accompagnées de ces vérifications et du comptage) seront répétées jusqu'à ce que le mot lu soit 'STOP' et que le casier Signal "soit" 'ROUGE'.

> Cette utilisation d'une variable "signal" (on dit aussi "drapeau" ("baissé" ou "levé") ou encore "sentinelle" ("endormie" ou "éveillée")) est fréquente en programmation. Elle est indispensable, chaque fois qu'un "évènement" doit "rester en mémoire" de l'exécutant amnésique.
> Nous avons choisi ici un Signal de type chaîne de caractères, plutôt qu'entier ou réel. Nous verrons dans la suite un type de casier se prêtant particulièrement bien à cet usage : le type booléen.

A côté du casier Signal, un casier MotLu (de type chaîne d'au plus 20 caractères) est évidemment indispensable (pour accueillir les mots fournis par l'utilisateur), ainsi que le traditionnel Compteur (de type entier).

Nous pouvons donc dresser la

Liste des variables

- MotLu, de type chaîne d'au plus 20 caractères
- Compteur, de type entier
- Signal, de type quelconque, pour autant qu'il puisse contenir une information au début et basculer en accueillant une autre information dans la suite.
  Ici, je l'ai choisi de type chaîne d'au plus 5 caractères

Ayant "réservé" les casiers indispensables, il me reste, comme d'habitude, à indiquer comment ils seront utilisés.

Marche à suivre

Il suffit de commander une répétition au cours de laquelle

- un mot nouveau sera lu et placé dans MotLu
- Compteur verra son contenu augmenter de 1
- le Signal passera à 'ROUGE' si MotLu contient 'ATTENTION'

> Lorsqu'on décrit la liste des instructions qui devront être répétées (on dit aussi "le corps de la boucle de répétition", il ne faut surtout pas la dresser en s'inspirant de la toute première fois que les actions correspondantes seront effectuées. Il faut en quelque sorte "prendre le train en marche" : on suppose, pour écrire le corps de la boucle que les actions sur lesquelles porte la répétition ont déjà été effectuées quelques fois et l'on écrit ce que sera ce corps lors d'une des fois où il sera répété. C'est ainsi que dans le texte ci-dessus on parle de "mot nouveau", supposant donc qu'il s'en est déjà présenté et donc que les actions à répéter

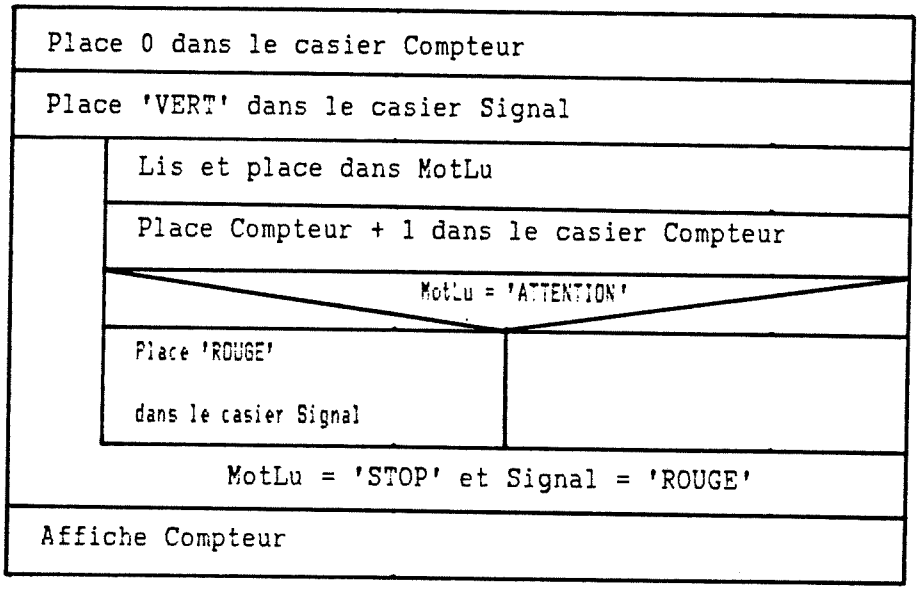
l'ont déjà été quelques fois. On s'intéresse ensuite à la condition d'arrêt de la boucle et aux actions qui doivent préparer la répétition (initialisations).
En d'autre termes, on écrira les actions à répéter au temps présent, tout en se disant que ces actions qu'on décrit ont déjà été réalisées dans le passé et qu'elles se poursuivront dans le futur (jusqu'à ce que la condition d'arrêt en stoppe la reprise).

Cette répétition s'interrompra lorsque le dernier mot lu (contenu de MotLu) sera 'STOP' et que Signal "sera" 'ROUGE'.

Comme toujours, il faudra commencer par les initialisations indispensables :

- Compteur doit primitivement contenir 0
- et Signal contenir 'VERT'

Nous sommes en mesure, dès lors de rédiger le GNS correspondant :



Remarques

(1) On a tout naturellemnt utilisé ici, dans l'énoncé de la condition de fin de répétition le mot "et". La condition énoncée ne sera vraie que si les deux comparaisons la constituant le sont. Cet outil (comme aussi "ou"), permettant de lier des comparaisons sont connus de l'exécutant et autorisent à énoncer des conditions plus ou moins élaborées. L'outil "non" (qui permet de nier une condition) est un peu du même genre.

(2) Retenons ici le "tour de main" de la variable-signal (on dit aussi "sentinelle" ou "drapeau"), fréquent en programmation. Il intervient à chaque fois qu'il est nécessaire de "faire retenir" qu'un évènement s'est produit.

----- O -----

Problème 3

La tâche est, on va le voir bien proche de la précédente.

Enoncé (Quoi faire ?)

Faire

lire successivement des mots (d'au plus 20 caractères),
arrêter à la donnée de STOP à condition que ATTENTION ait été fourni juste avant.
Dire alors combien de mots ont été lus.

Les étapes essentielles de la démarche ont été longuement illustrées et commentées sur les deux exemples précédents. Je me permettrai à présent d'être un peu plus laconique.

Comment faire ?

1) On retient un instant chaque mot lu, en étant particulièrement attentif au mot suivant lorsque c'est ATTENTION qui vient d'être lu. Si le mot suivant n'est pas STOP, alors on "oublie" que ATTENTION venait d'être lu. De plus, on compte tous les mots.

ou encore

2) On retient un instant le dernier mot lu et l'avant-dernier. On compte au fur et à mesure.

Comment faire faire ?

Première solution

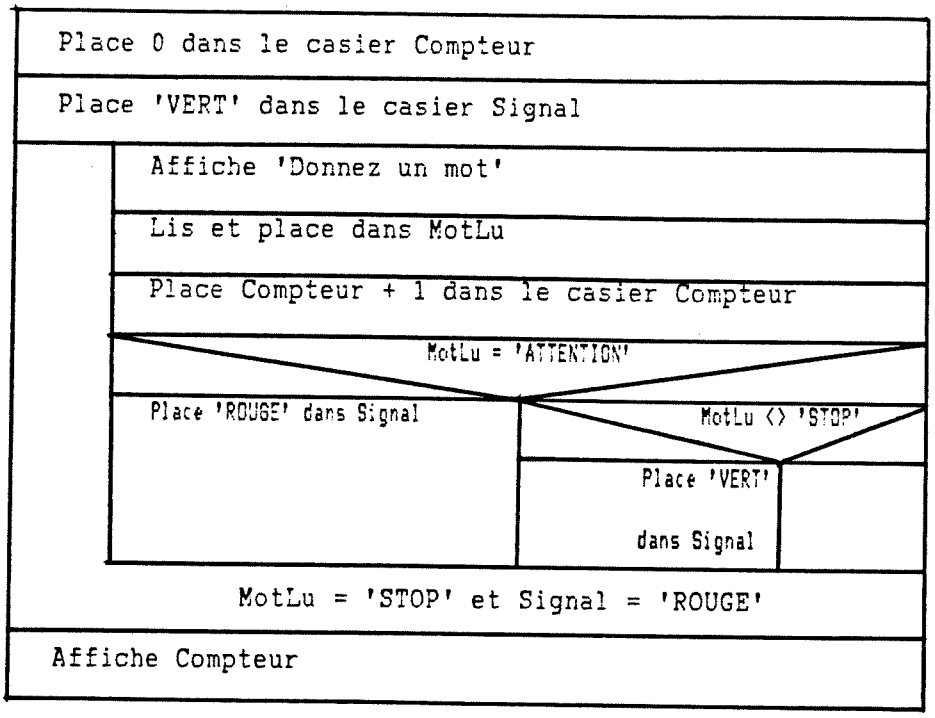
Je vais faire usage, à nouveau, d'un signal qui détectera le passage de 'ATTENTION'. Signal passera à 'ROUGE' à la lecture de 'ATTENTION'. Mais ce 'ROUGE' ne compte que si, à la lecture suivante, c'est le mot 'STOP' qui est fourni. Sinon, Signal doit repasser au 'VERT'. Autrement dit, lorsque le mot lu n'est pas 'ATTENTION' et que le signal est 'ROUGE' (donc 'ATTENTION' vient d'être lu), il est impératif qu'il repasse à 'VERT' si le mot lu n'est pas 'STOP'. (Ouf ...!)

Cette approche, qui privilégie une nouvelle utilisation de variable signal est, on en conviendra assez complexe, pour ne pas dire obscure. On le verra, l'approche suivante semble nettement plus simple et "naturelle".

Liste des variables

- Motlu, de type chaîne d'au plus 20 caractères
- Compteur, de type entier
- Signal, de type chaîne d'au plus 5 caractères, passera au 'ROUGE' si 'ATTENTION' est lu, mais en restant 'ROUGE' seulement si c'est 'STOP' qui suit; sinon, il repasse au 'VERT'.

Marche à suivre



Deuxième Solution

Elle correspond à l'attitude consistant à retenir chaque mot, ainsi que le précédent, tout en les comptant. Trois variables seront dès lors nécessaires : celle contenant le dernier mot lu, celle contenant le précédent et celle permettant de compter.

Liste des variables

- Dernier, de type chaîne d'au plus 20 caractères, qui contiendra le dernier mot lu
- AvantDernier, de type chaîne d'au plus 20 caractères, qui contiendra l'avant dernier mot
- Compteur, de type entier

Marche à suivre

La seule difficulté, c'est de faire en sorte qu'au "bon moment", le contenu de Dernier soit transféré dans AvantDernier.

Après les initialisations d'usage, je commanderai de répéter un même groupe d'actions. On va voir que l'ordre dans lequel ces actions sont commandées est, ici, fort important. Si je commande de répéter

- lire un mot et le placer dans Dernier
- recopier Dernier dans AvantDernier
- compter

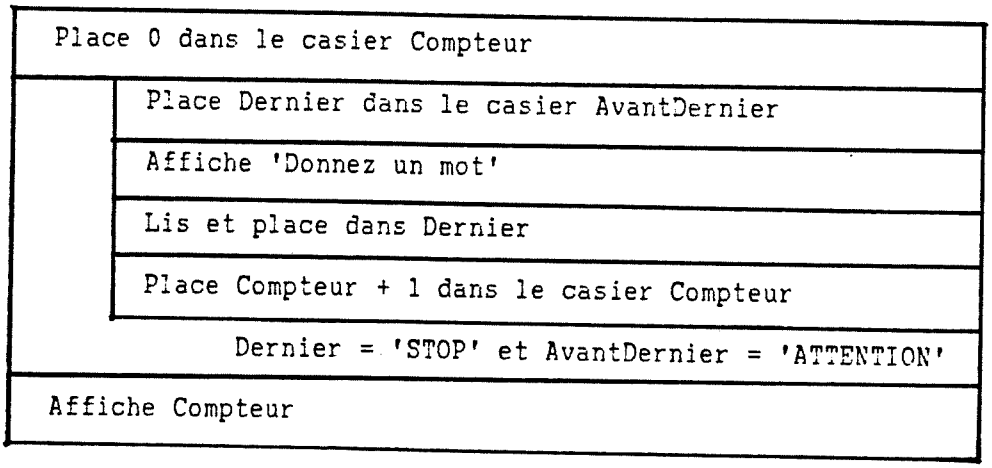
et cela jusqu'à ce que Dernier contienne 'STOP' et AvantDernier contienne 'ATTENTION', j'aboutis à une absurdité puisque Dernier et AvantDernier contiendront à coup sûr le même mot.

En réalité, il faut d'abord faire transférer le contenu de Dernier dans AvantDernier AVANT de faire lire un nouveau mot pour le placer dans Dernier (et non après, comme ci-dessus).

Je commanderai donc la répétition des actions suivantes (cette fois dans le bon ordre) :

- recopier Dernier dans AvantDernier
- lire un mot et le placer dans Dernier
- compter

Avec l'habituelle initialisation du Compteur, ceci conduit au GNS :



Malheureusement, un détail rend ce programme incorrect. Imaginons un instant que par un improbable et malheureux hasard, le casier Dernier contienne, avant le début des opérations le mot 'ATTENTION' et que le tout premier mot fourni par l'utilisateur soit 'STOP'. L'exécutant va successivement poser les actions suivantes (commandées par le GNS):

- mettre 0 dans Compteur
- recopier le contenu de Dernier (il y "traîne" 'ATTENTION') dans AvantDernier, qui contiendra alors, lui aussi, 'ATTENTION'
- lire le premier mot ('STOP') et le placer dans Dernier
- augmenter le contenu de Compteur, qui passe donc à 1
- voir si Dernier contient bien 'STOP' (c'est le cas) et AvantDernier contient 'ATTENTION' (c'est aussi le cas) et donc interrompre immédiatement la répétition
- afficher 1, contenu de Compteur et arrêter.

Il est facile de voir qu'il faut à tout prix éviter que Dernier contienne primitivement 'ATTENTION', et prendre la peine de le faire initialiser en y plaçant par exemple 'BONJOUR'.

> Ceci nous enseigne qu'il faut obligatoirement être fort attentif à ce problème d'initialisation des casiers. Lorsque la première manipulation commandée à propos d'un casier consiste à en prendre le contenu (comme ce serait le cas ici, si je n'avais pas fait les initialisations, pour Dernier et Compteur), il faut impérativement réfléchir à l'importance de son contenu primitif. Par contre, lorsque la première manipulation faisant intervenir un casier commande d'y placer une information, son initialisation est, généralement, inutile.

Cette précaution conduit au GNS :

Place 0 dans le casier Compteur
Place 'BONJOUR' dans le casier Dernier
Place Dernier dans le casier AvantDernier
Affiche 'Donnez un mot'
Lis et place dans Dernier
Place Compteur + 1 dans le casier Compteur
Dernier = 'STOP' et AvantDernier = 'ATTENTION'
Affiche Compteur

----- 0 -----

Problème 4

Enoncé (quoi faire ?)

Faire

Simuler des jets successifs d'un dé (en montrant les résultats) et en arrêtant au troisième six obtenu.
Indiquer alors combien de "lancers" ont été nécessaires.

> Simuler signifie ici "faire comme si". Le programme à rédiger doit faire en sorte qu'apparaissent à l'utilisateur une succession de nombres, compris entre 1 et 6 (les résultats des "lancers"). Cette liste doit s'interrompre à l'apparition du troisième 6 et être alors suivie du nombre total de chiffres apparus.
>
> Cet énoncé n'a évidemment de sens que si l'exécutant dispose d'un outil permettant de tirer des nombres au hasard.

Disons simplement qu'on pourra se permettre d'écrire l'instruction d'affectation :

Place un nombre au hasard entre 1 et 6 dans ...

Je me permets donc de désigner une information entière à manipuler par

un nombre (entier) au hasard entre 1 et 6

en indiquant comment ceci sera traduit lors du codage en Pascal.

Comment faire ?

Ce qui importe, une fois de plus, c'est de percevoir ce que je retiendrais comme informations (susceptibles) de se modifier) si c'était moi qui lancais le dé.

Je garderais à l'esprit :

- un instant chaque résultat, pour l'annoncer (le faire connaître à l'utilisateur)
- le décompte de tous les lancers, puisqu'il faudra, à la fin, pouvoir indiquer combien il y en a eu
- le nombre de six déjà obtenus, pour être en mesure d'arrêter au troisième.

> Ces informations retenues vont à nouveau donner naissance aux casiers nécessaires lorsqu'il s'agira de s'intéresser à l'exécutant "gestionnaire de casiers".

Comment faire faire ?

Nous pouvons donc immédiatement dresser la

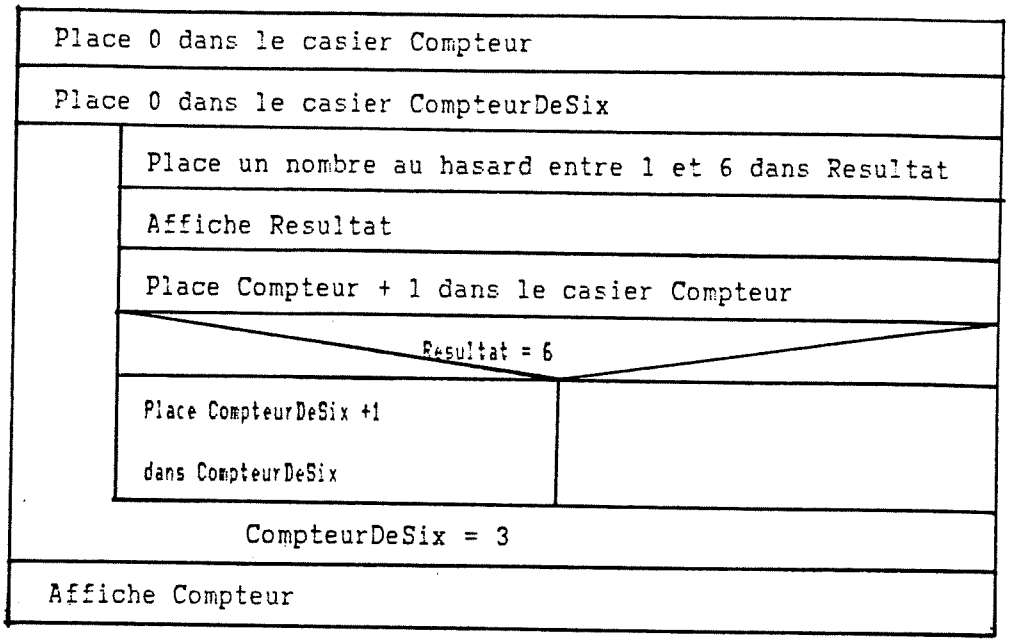
Liste des variables

Resultat, de type entier, qui contiendra un instant chaque résultat du tirage au hasard
Compteur, de type entier, pour compter les lancers; il s'augmentera de 1 à chaque lancer
CompteurDeSix, de type entier, pour compter les six qui se présenteront.

> On remarquera, que pour préparer la traduction en Pascal, j'ai, dès à présent choisi de noter "Resultat" (sans accent) la variable comportant les résultats des tirages. En effet, les noms de variables ne pourront, en Pascal, comporter de lettres accentuées.

Marche à suivre

Son écriture est immédiate : je vais à nouveau commander une répétition au cours de laquelle un "tirage" sera effectué, affiché, compté et comptabilisé (dans CompteurDeSix) s'il s'agit d'un 6. Cette répétition s'arrêtera lorsque le CompteurDeSix contiendra 3.



----- 0 -----

Problème 5

Enoncé (quoi faire ?)

Faire

lire 100 données réelles et
fournir ensuite leur moyenne, la plus petite et la plus grande.

Voici enfin une tâche un peu moins gratuite et un peu plus utile. Cette utilité est cependant gravement réduite par le fait que le programme réalisé ne pourra traiter qu'une succession d'exactement 100 données, ni plus ni moins. Pas question, dès lors de s'en servir pour calculer la moyenne d'une série quelconque de données.

Un exercice (Cf ci-après) tentera d'étendre ces spécifications pour obtenir un produit réellement utilisable.

Comment faire ?

Que retiendrons-nous, face à ce travail et, première question, sommes-nous forcés de garder en tête, simultanément les 100 données à traiter ?

En réalité, chacune des données ne sera retenue qu'un court instant, le temps de la traiter. Ce traitement de la donnée reçue consistera d'abord à la compter (puisqu'il faut arrêter à la centième qui sera fournie); ensuite, pour pouvoir à la fin évaluer la moyenne, il suffit qu'au fur et à mesure je retienne la somme des données déjà acquises : je mettrai donc cette somme à jour en y ajoutant à chaque fois la dernière donnée acquise; enfin, à tout moment je garde aussi en mémoire quelle a été

jusqu'alors la plus petite et la plus grande donnée reçue : chaque donnée nouvelle sera comparée à ces deux quantités et elles s'en trouveront éventuellement modifiées. A la fin, il suffira de diviser la somme (obtenue au fur et à mesure) par 100 pour obtenir la moyenne.

Ainsi donc, je retiendrais

- chaque donnée, le temps de la traiter
- le décompte des données reçues
- la somme des données obtenues
- la plus petite des données
- la plus grande

Comment faire faire ?

Nous pouvons immédiatement dresser la

Liste des variables

- Donnee, de type réel, qui contiendra chaque donnée lue,
- Compteur, de type entier, qui permettra de les compter,
- Somme, de type réel, qui accueillera la somme des données,
- PlusPetite, de type réel, qui contiendra la plus petite donnée,
- PlusGrande qui contiendra la plus grande.

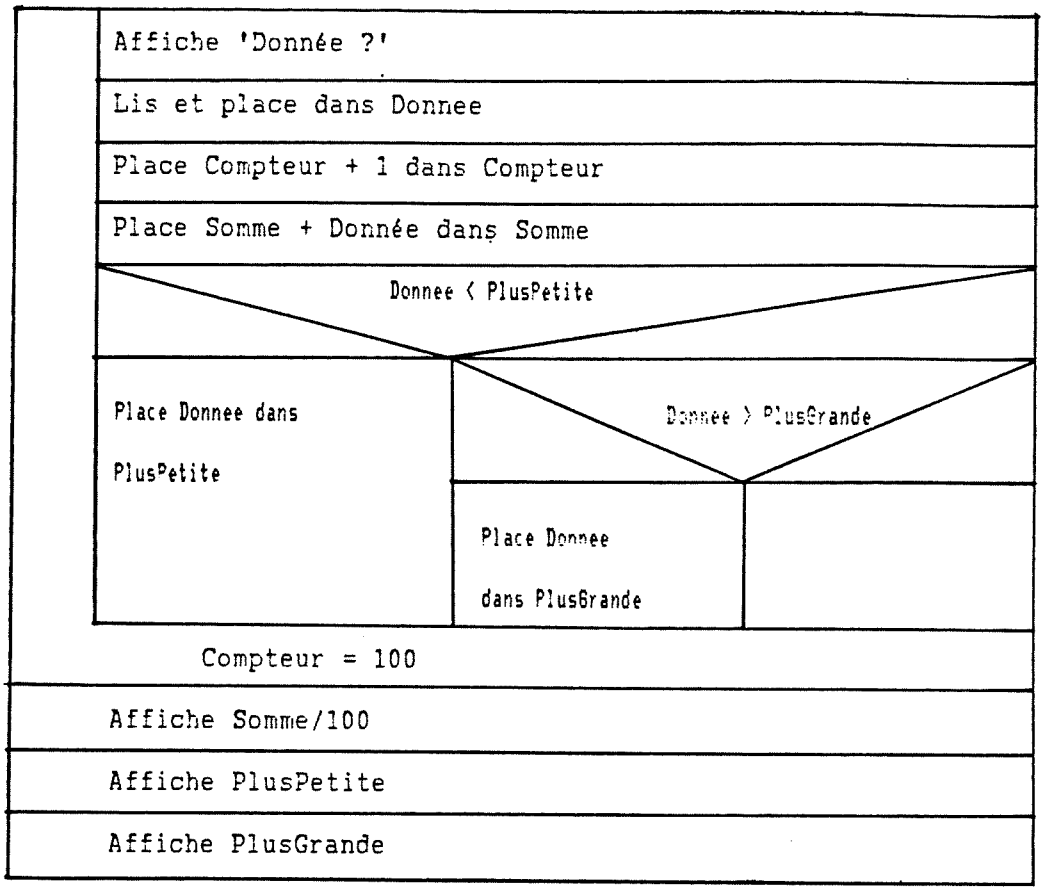
Marche à suivre

Le coeur de celle-ci sera à nouveau constituée d'une répétition qui comportera

- la lecture d'une donnée (déposée) dans Donnee
- l'augmentation du casier Compteur permettant de les compter
- l'ajout de la Donnée à la Somme
- la comparaison de Donnée à PlusPetite et à PlusGrande et la mise à jour éventuelle d'un de ces casiers.

Cette répétition s'interrompra lorsque le Compteur signalera que la 100ème donnée à été acquise et traitée.

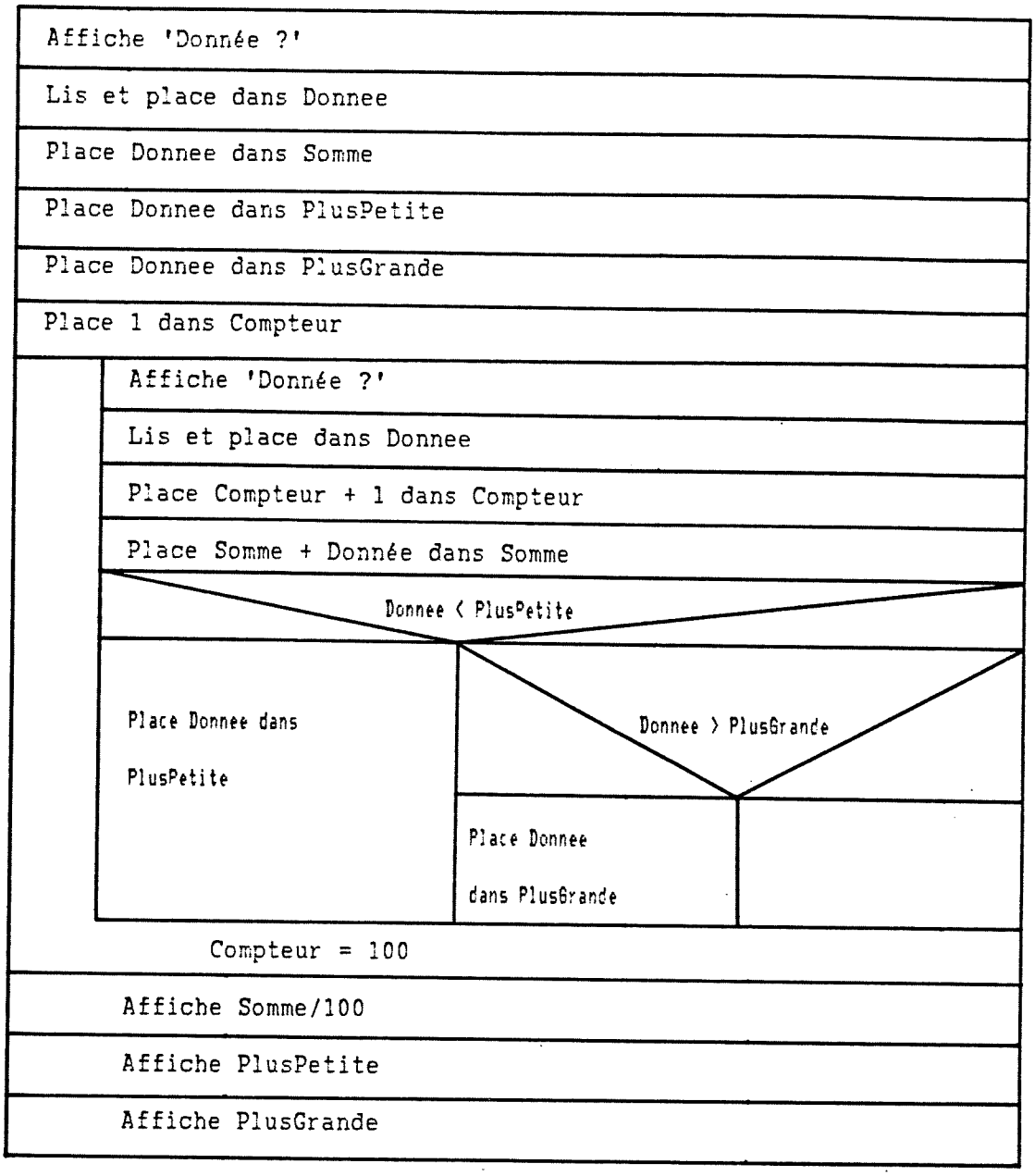
Ceci conduit au GNS



Reste évidemment à traiter le problème des initialisations. On voit aisément que Compteur et Somme doivent être placés à 0 avant le début des répétitions. Le cas des casiers PlusPetite et PlusGrande est plus délicat. Ainsi, si je fais placer primitivement 0 dans PlusPetite et si la plus petite des données fournies est 15, il sera répondu, à la fin que la plus petite est 0 (valeur primitivement contenue dans PlusPetite et inférieure à toutes les données reçues. En effet, à la question "Donnee < PlusPetite", il sera toujours répondu "non" et le contenu primitif de PlusPetite restera inchangé. On voit donc, qu'en réalité, il faudrait faire placer dans PlusPetite une quantité à coup sûr supérieure aux données qui seront fournies. On pourrait y placer une valeur très grande (1000000 ou plus); mais ceci conduirait à un programme erroné dans le cas bien improbable ou toutes les données reçues seraient plus élevées que cette quantité.
Le même problème se pose évidemment pour la détection de la plus grande des données.

En réalité, une approche un peu différente permet de venir à bout de cette difficulté: il suffit de traiter un peu différemment la toute première donnée qui peut servir à initialiser Somme, PlusPetite et PlusGrande; en n'oubliant pas de placer 1 dans le Compteur, puisque la première donnée faisant l'objet d'un traitement spécifique, hors de la boucle de répétition, on commencera celle-ci avec une donnée déjà lue et traitée.

Ceci conduit alors au GNS :


Affiche 'Donnée ?'
Lis et place dans Donnee
Place Donnee dans Somme
Place Donnee dans PlusPetite
Place Donnee dans PlusGrande
Place 1 dans Compteur
Affiche 'Donnée ?'
Lis et place dans Donnee
Place Compteur + 1 dans Compteur
Place Somme + Donnée dans Somme
Donnee < PlusPetite
Place Donnee dans
PlusPetite
Donnee > PlusGrande
Place Donnee
dans PlusGrande
Compteur = 100
Affiche Somme/100
Affiche PlusPetite
Affiche PlusGrande

EXERCICES

I) Pour chacune des tâches suivantes, effectuer les diverses étapes :

- Comment vous y prendriez vous, aux prises avec ce travail, en étant essentiellement attentif aux informations que vous retiendriez ?
- Quelles sont les variables indispensables, quel est leur type et quel en sera le contenu ?
- Quelle est la marche à suivre, exprimée sous forme GNS?

FAIRE

1) Lire successivement 10 nombres réels et signaler ensuite combien il y en avait de négatifs.

2) Demander d'abord à l'utilisateur combien de données réelles devront ensuite être lues; lire successivement ces données et fournir ensuite le nombre de données strictement négatives, le nombre de données strictement positives et le nombre de données nulles.

3) Simuler des lancers successifs d'une pièce de monnaie jusqu'à ce que le nombre de "pile" dépasse de 3 le nombre de "face" et fournir alors le nombre de lancers qui auront été nécessaires pour cela.
On peut utiliser l'affectation

Place au hasard, soit 1, soit 2 dans ....

4) Simuler 1000 lancers successifs d'une pièce de monnaie en fournissant ensuite le nombre de "pile" et le nombre de "face" obtenus.

5) Simuler des lancers successifs d'un dé en montrant chacun des résultats et en s'arrêtant au troisième six obtenu. Signaler ensuite en quelle position se trouvait le premier 6 obtenu, en quelle position se trouvait le second, et en quelle position se trouvait le troisième.

6) Lire une valeur entière et positive N; calculer et afficher alors N! (factorielle de N).
Rappel : N! = 1*2*3*4* ... *N

7) Simuler des jets successifs d'un dé (en montrant les résultats) en arrêtant lorsqu'un "cinq" est immédiatement suivi d'un "six". Fournir alors le nombre de "un" obtenus.

8) Lire successivement des nombres réels en s'arrêtant lorsque 0 est fourni. Donner alors la somme des nombres positifs et celle des nombres négatifs.

9) Demander à l'utilisateur combien de données entières seront à lire; procéder à la lecture de ces données en précisant ensuite combien de fois il s'en est présenté qui soient identiques à la première.

10) Lire des nombres entiers en arrêtant lorsque 0 est fourni; signaler alors combien de fois il s'est présenté deux 1 successifs. Par exemple, si la succession des nombres est

1 4 1 1 2 1 1 1 0

on détectera trois paires de 1 successifs.

11) Lire des nombres entiers en arrêtant à la troisième fois que le nombre 1 est fourni et indiquer alors combien de fois le nombre 0 etait présent dans la succession.

12) Vérifier la connaissance des tables de multiplication de l'(élève) utilisateur. Pour cela, proposer successivement 20 énoncés du type

... x ... = ?

les nombres y intervenant étant choisis au hasard entre 2 et 10. A chaque énoncé, lire la réponse fournie par l'utilisateur. Si cette dernière est correcte, le féliciter; si elle est fausse, le signaler et laisser une seconde chance; si elle est fausse à nouveau, donner la réponse correcte. A la fin, fournir le nombre de bonnes réponses obtenues directement (sans second essai).

13) Lire successivement des mots (d'au plus 20 caractères) en arrêtant à la donnée de STOP à condition que ATTENTION ait été fourni n'importe quand auparavant. Donner alors la position du premier et du dernier ATTENTION apparu.

14) Demander à l'utilisateur par quelle nombre il terminera la liste des données réelles qu'il fournira. Lire alors successivement ces données (jusqu'à l'apparition de la donnée marquant la fin). Préciser ensuite combien de données n'étaient pas comprise entre -10 et 10. Attention, la dernière donnée provoquant l'arrêt ne doit en aucun cas être comptabilisées.
Remarque : le programme ne doit pas "se planter" même si la seule donnée fournie par l'utilisateur (idiot) est justement celle qu'il avait choisie pour provoquer l'arrêt.

----- 0 -----

II) J'ai signalé, en ce qui concerne le problème 5 (calcul de la moyenne) que la restriction à 100 données rendait le programme résultant peu utilisable.
Comment pourrait-on changer les spécifications pour s'affranchir de cette limitation en obtenant un programme qui puisse traiter un nombre de données quelconque ? Peut-on faire en sorte que l'utilisateur ne soit pas tenu de compter les données avant de commencer à les fournir ?
Pour chacun des énoncés auxquels conduiront ces extensions, exposer la démarche de programmation suivie en allant jusqu'à l'écriture du GNS.

----- 0 -----

III) Les marches à suivre correspondant aux problèmes abordés ont essentiellement comporté des répétitions sous la forme Répéter ... jusqu'à ce que ....Pouvez- vous réécrire les GNS correspondants en utilisant la structure Tant que ... faire ...

----- 0 -----

CHAPITRE 5

TRADUCTION EN PASCAL

Avant de se retrouver (enfin) assis face à l'ordinateur, l'avant-dernière étape du traitement informatique d'une tâche est celle au cours de laquelle la marche à suivre obtenue sous forme GNS sera traduite dans un langage de programmation.

Cette étape du "Comment dire ?" ne réclame plus guère d'imagination et de recherche. Simplement, il s'agit à présent de respecter scrupuleusement les règles de grammaire et d'orthographe imposées par le langage compromis. Rien d'essentiel ne sera ajouté au contenu des GNS : nous les exprimerons différemment et avec moins de liberté quant à la syntaxe et à l'orthographe. Le seul avantage de ce type d'expression de la marche à suivre, c'est qu'il sera "compris par l'ordinateur". Cette étape est donc à la fois indispensable (pour finir par essayer vraiment les marches à suivre conçues) et fastidieuse, puisqu'il ne s'agit plus du tout de faire preuve d'invention ou de recherche, mais seulement de docilité et de rigueur formelle.

Chacun des exemples abordés et traités dans le chapitre précédent va maintenant être repris et la traduction en Pascal sera indiquée et commentée. Le traitement de ces exemples permettra évidemment de découvrir les premiers rudiments du langage Pascal.

Enfin, une synthèse reprendra, en fin de chapitre, de manière structurée, les éléments qui, peu à peu, auront été découverts.

Problème 1

Il s'agissait, rappelons-le, de faire lire une série de nombres réels, jusqu'à ce que 0 apparaisse et de fournir alors le décompte de ces données.
Notre analyse avait débouché sur le résultat suivant :

Liste des variables (casiers)

NombreFourni, de type réel, qui contiendra chaque donnée lue
Compteur, de type entier, pour les compter

Marche à suivre

Place 0 dans le casier Compteur
Affiche 'Donnez un nombre'
Lis et place dans NombreFourni
Place Compteur + 1 dans le casier Compteur
NombreFourni = 0
Affiche NombreFourni

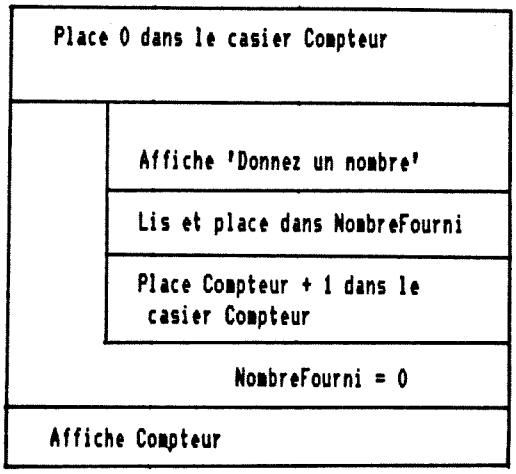
3055

Dans un premier temps, et afin de montrer clairement qu'il s'agira (bêtement) d'exprimer différemment les structures de contrôle et les actions élémentaires présentes dans le GNS, j'ai fait figurer côte à côte ce dernier et son expression en Pascal.

Liste des variables (casiers)

NombreFourni, de type réel,
Compteur, de type entier

Marche à suivre




```
program ARRETA0;

var

     NombreFourni : real ;
     Compteur : integer ;

begin

Compteur := 0;

repeat

     writeln('Donnez un nombre');

     readln(NombreFourni);

     Compteur := Compteur +1;

until NombreFourni = 0;

writeln (Compteur);

end .
```

Revoici sous une forme plus lisible le texte du programme Pascal mis en évidence. Il sera suivi d'un certain nombre de remarques explicatives.

```
program ARRETA0;                    (1)   <1>

var NombreFourni : real ;           (2)
    Compteur : integer ;            (3)

begin                               (4)
Compteur := 0;                      (5)
repeat                              (6)
     writeln('Donnez un nombre');   (8)
     readln(NombreFourni);          (7)
     Compteur := Compteur +1;
until NombreFourni = 0;             (6)
writeln (Compteur);                 (8)
end .                               (4)
```

<1> *Dans ces notes, les mots réservés du Pascal seront toujours soulignés et apparaîtront en gras à leur première rencontre. Il n'en sera pas de même dans les textes de programmes que vous aurez l'occasion de fournir à l'ordinateur : aucun mot n'y sera particulièrement mis en évidence.*

Remarques

(1) program ARRETA0;

Tout programme Pascal commence par un entête. Le mot réservé program est suivi de l'identification de celui-ci (ici ArretA0). Cet identificateur du programme doit commencer par une lettre, ne comporter que des lettres (majuscules ou minuscules),le symbole de soulignement (_) et des chiffres. La même règle s'appliquera à tous les identificateurs rencontrés dans la suite (par exemple ceux désignant les variables ou les procédures).

(2) var NombreFourni : real ;
Compteur : integer ;

La deuxième zone d'un programme Pascal est la partie déclaration. Pour l'instant, cette zone comporte seulement la liste des variables (casiers) utilisés. Cette liste s'annonce par le mot réservé var (qui n'est écrit qu'une seule fois). Vient ensuite la liste des variables des divers types : integer, real,...

(3) On notera que les diverses listes de variables sont terminées par le ;. Les identificateurs des variables suivent les règles énoncées ci-dessus. Nous les avions d'ailleurs respectées lors du choix des noms de casiers à l'étape précédente.

(4) Vient ensuite le corps du programme, écrit entre les mots *begin* et *end*, ce dernier étant suivi du point final. Ce corps traduit la marche à suivre telle qu'elle est présentée dans le GNS.

(5) Compteur := 0;

L'instruction d'affectation est traduite par le symbole :=, le nom de la variable à remplir étant écrit à gauche, l'information à y placer figurant à droite.

(6) repeat ... until NombreFourni = 0;

La structure de répétition Répéter ... jusqu'à ce que *condition* est traduite par repeat liste d'instructions until *condition*.

(7) readln(NombreFourni);

L'instruction de lecture est traduite par readln suivi par le nom de la variable où déposer l'information lue, placé entre parenthèses.

(8) writeln (Compteur);

L'instruction d'affichage est traduite par writeln suivi par l'information (ou la liste des informations) à afficher, entre parenthèses.

(9) J'y reviendrai en détail dans la suite, mais dès à présent, il faut remarquer que le symbole qui permet de séparer les diverses instructions Pascal est le point-virgule. Les passages à la ligne sont syntaxiquement sans importance (en général) de même que les retraits de certaines lignes.

Au delà de la syntaxe à respecter, faute de quoi des erreurs seront détectées lors de la compilation, il faut aussi être attentif à rédiger un programme clair, bien présenté et (donc) facilement lisible. J'y reviendrai ci-dessous en parlant de l'attention à porter au lecteur du programme.

Nous voici presqu'au terme de notre démarche. Il nous reste seulement à "habiller" le programme obtenu.

"Habiller" un programme, ce sera d'abord être attentif à l'utilisateur. L'utilisateur, rappelons-le, dans notre terminologie, c'est l'être humain qui se trouve de l'autre côté de l'écran et du clavier pendant que l'exécutant effectue les actions commandées par notre marche à suivre. C'est lui qui lira à l'écran les informations dont nous demanderons l'affichage; c'est lui qui passera à travers le clavier les informations que l'exécutant attendra lors des instructions de lecture.

Et, dans l'état actuel du programme, l'utilisateur n'est absolument pas averti des objectifs de ce programme, des données qui sont attendues ni des résultats qui seront fournis. Tout au plus voit-il apparaître l'invitation "Donnez un nombre", correspondant à l'affichage que nous commandons.

En tant que programmeurs, nous ne sommes évidemment pas en contact direct avec l'utilisateur; nous avons, en quelque sorte, délégué à l'exécutant nos prérogatives en lui fournissant la marche à suivre. C'est donc seulement à travers les instructions fournies à cet exécutant que nous pouvons nous préoccuper, en différé, de l'utilisateur.

La seule manière d'être attentif à ce dernier consiste à commander l'affichage de messages explicatifs qui lui indiqueront le traitement que permet le programme, les données qui sont attendues et les résultats qui seront fournis. Nous allons donc ajouter au programme existant les instructions d'affichage indispensables.

Ceci conduit à la version suivante, où les instructions supplémentaires sont écrites en italique :

```
program ARRETA0;

var NombreFourni : real ;
    Compteur : integer ;

begin
writeln('Vous allez me fournir des nombres réels en faisant');     (1)
writeln('suivre chacun d''eux de l''appui sur la touche d''entrée.'); (2)
writeln('Vous terminerez la série par 0 et je vous dirai alors combien');
writeln('de nombres vous m''avez fourni au total.');
Compteur := 0;
repeat
     write('Donnez un nombre : ');      (3)
     readln(NombreFourni);
     Compteur := Compteur +1;
until NombreFourni = 0;
writeln ('Nombre de données fournies : ',Compteur);  (4)
end .
```

Remarques

(1) Les "messages" dont nous commandons l'affichage ne sont bien entendu rien d'autre pour l'exécutant que des constantes de type chaîne de caractères. Il faut savoir qu'une telle constante peut comporter n'importe quel caractère sauf celui indiquant un passage à la ligne. Nous ne pouvons donc pas écrire

writeln('Vous allez me fournir des nombres réels en faisant suivre chacun ...

C'est pourquoi chaque nouvelle ligne de message demande une nouvelle instruction d'affichage, d'où la multiplication des writeln.

(2) Lorsqu'une constante chaîne de caractères (qui est toujours enclose entre des apostrophes) comporte le caractère apostrophe, on est tenu de redoubler celui-ci. Lors de l'affichage, il ne sera écrit qu'une seule fois. Attention, il ne faut pas confondre l'apostrophe redoublé ('') et les guillemets (").

(3) Nous avions déjà, dans la version "nue" du programme fait figurer un affichage sommaire invitant l'utilisateur à fournir un nombre. J'ai quelque peu modifié et enjolivé cette instruction, essentiellement en remplaçant writeln par write. L'instruction writeln provoque un passage à la ligne à la suite des informations affichées. L'instruction write ne commande pas ce passage à la ligne. Ainsi, pendant l'exécution du programme "habillé" l'écran se présentera de la façon suivante :

```
Vous allez me fournir des nombres réels en faisant
suivre chacun d'eux de l'appui sur la touche d'entrée.
Vous terminerez la série par 0 et je vous dirai alors combien
de nombres vous m'avez fourni au total.
Donnez un nombre : 12.5
Donnez un nombre : -2
Donnez un nombre : 0
Nombre de données fournies : 3
```

les informations soulignées étant celles fournies au clavier par l'utilisateur.

(4) Nous commandons l'affichage de deux informations qui se suivront, collées l'une à l'autre, sur la même ligne. La première est la constante chaîne de caractères 'Nombre de données fournies : ' (et l'on remarquera l'espace qui la termine); la seconde est le contenu du casier Compteur.

Ce premier type d'habillage étant terminé, reste à indiquer ce que sera le second. Il ne s'adressera plus à l'utilisateur mais au lecteur du texte même du programme. Il s'agit là (comme pour l'utilisateur) de l'être humain, connaissant généralement la programmation, et à qui nous transmettons pour information tout le dossier de programmation d'une application (du "Quoi Faire ?" au "Comment dire ?"). Le texte du programme est la dernière pièce de ce dossier et nous y ferons donc figurer des commentaires (remarques) pour aider ce lecteur à comprendre l'analyse que nous avons menée et le programme qui en a résulté.

Ainsi, nous rappellerons les objectifs du programme construit; pour chacune des variables, nous indiquerons à quoi elle servira (les informations qui y seront contenues), ...

Cela conduira au programme suivant :

```
program ARRETA0;
(* Ce programme fait lire des nombres réels jusqu'à ce que 0 soit fourni  et fait alors afficher combien de
nombres ont été lus. *)

var NombreFourni
     (* qui contiendra les données lues *)
     : real ;
    Compteur
    (* pour compter les données *)
     : integer ;

begin
(* Avertissement de l'utilisateur *)
writeln('Vous allez me fournir des nombres réels en faisant');
writeln('suivre chacun d''eux de l''appui sur la touche d''entrée.');
writeln('Vous terminerez la série par 0 et je vous dirai alors combien');
writeln('de nombres vous m''avez fourni au total.');
(* Initialisation du compteur *)
Compteur := 0;
repeat
     write('Donnez un nombre : ');
     readln(NombreFourni);
     Compteur := Compteur +1;
until NombreFourni = 0;
writeln ('Nombre de données fournies : ',Compteur);  (4)
end .
```

Les commentaires sont notés en caractères condensés. On voit qu'ils sont enclos dans les symboles (* *). On pourrait aussi employer { }. Ces commentaires sont bien entendu ignorés de l'exécutant et ne seront pas visibles pour l'utilisateur. Ils peuvent être introduits n'importe où dans le texte du programme (sauf évidemment en plein milieu d'un mot réservé ou d'un identificateur); ce ne sont pas des instructions et, dès lors, il est inutile de les faire suivre par le point-virgule.

A côté des commentaires qui explicitement sont là pour faciliter la lecture et la compréhension du programme, il faut noter la présentation utilisée : passages à la ligne, indentation (retrait) de certaines parties du programme (par exemple le corps de la boucle de répétition), passage de lignes blanche, ... Pas de règle syntaxique très stricte à ce niveau : du bon sens en tant que rédacteur et la volonté de faciliter la tâche de celui qui aura à relire et à comprendre le programme conçu.

Dans le même ordre d'idée, nous avons pris la bonne habitude de choisir des noms de variables "parlant". Rien n'est plus pénible que ces variables X, Y, T, C ... dont l'étiquette n'annonce en rien le contenu!

> Cette préoccupation de lisibilité est sans doute un peu inutile pour des programmes qui font seulement quelques lignes. Elle deviendra impérative dès que les exemples traités deviendront plus complexes et déboucheront sur des programmes plus longs. Faute de cette volonté d'être clair et lisible, les textes des programmes deviennent rapidement une jungle dans lequel le lecteur se perd ... y compris d'ailleurs lorsque le lecteur est le concepteur-même du programme qui essaye tout bêtement de se relire.

> C'est pour cette raison que dès à présent je vous recommande de prendre ces bonnes habitudes de rédaction et de présentation. Il vaut mieux avoir adopté cette attitude lorsqu'elle est encore accessoire et "facultative" que de l'ignorer lorsqu'elle devient indispensable. Nul ne saura jamais les milliers de programmes qui ont été mis à la poubelle faute qu'on puisse simplement les relire pour les modifier et les amender.
> La programmation, c'est essentiellement de la "matière grise ajoutée": il faut veiller à ce qu'elle soit facilement perçue et non la dissimuler dans des textes de programmes obscurs et illisibles.

Nous voilà au terme du traitement de ce premier exemple. Nous y avons illustré l'ensemble de la démarche (de la définition précise de la tâche à l'habillage du programme Pascal). Cette démarche restera inchangée, en tout cas en ce qui concerne ce chapitre. Simplement, je me permettrai d'être nettement plus succint en ce qui concerne les exemples suivants.

----- o -----

Problème 2

Il s'agissait, cette fois de (faire) lire et compter des mots avec arrêt au mot STOP à condition que ATTENTION soit survenu auparavant.

Première solution

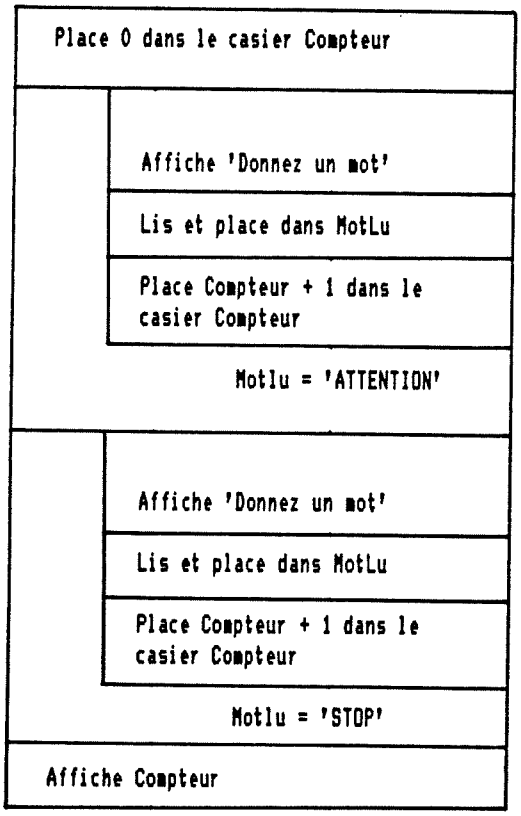
C'était celle où deux répétitions successives étaient commandées, la première s'interrompant à la lecture de ATTENTION, la seconde à celle de STOP.

Voici à nouveau, côte à côte le GNS (tel que nous l'avions conçu dans l'étape précédente) et son expression en Pascal :

Liste des variables

MotLu, de type chaîne d'au plus 20 caractères
Compteur, de type entier

Marche à suivre




```
program ATTENTION1;

var

     MotLu : string[20] ;
     Compteur : integer ;

begin

Compteur := 0;

repeat

     write ('Donnez un mot');

     readln(MotLu);

     Compteur := Compteur +1;


     until MotLu = 'ATTENTION';

repeat

     write ('Donnez un mot');

     readln(MotLu);

     Compteur := Compteur +1;


until MotLu = 'STOP';

writeln(Compteur);

end.
```

Revoici d'ailleurs le texte de ce programme assorti des habituelles remarques :

```
program ATTENTION1;

var MotLu : string[20] ;            (1)
    Compteur : integer ;

begin
Compteur := 0;
repeat
     readln(MotLu);
     Compteur := Compteur +1;
until MotLu = 'ATTENTION';          (2)(3)
repeat
     readln(MotLu);
     Compteur := Compteur +1;
until MotLu = 'STOP';               (2)
writeln(Compteur);
end.
```

Remarques

(1) `var MotLu : string[20] ;`

Le type "chaîne d'au plus 20 caractères" se traduit en Pascal par `string[20]`. Bien évidemment, il est possible de définir des variables chaînes de caractères en comportant un nombre différent : il suffit de faire figurer ce nombre entre crochets à la suite du mot `string`; il s'agit toujours du nombre maximal de caractères que pourra comporter une chaîne accueillie dans cette variable. La limite de ce nombre maximal de caractères est cependant fixée à 255.

(2) `until MotLu = 'ATTENTION';`

Nous avions jusqu'à présent fait comparer des nombres à l'aide du symbole =. On peut aussi faire comparer des informations de type chaîne de caractères.

(3) Les constantes de type chaîne de caractères doivent, rappelons-le une fois de plus, être écrites entre apostrophes.

Il reste à présent à habiller ce programme en tenant compte de l'utilisateur :

```
program ATTENTION1;

var MotLu : string[20] ;
    Compteur : integer ;

begin
writeln('Vous allez me fournir successivement des mots');
writeln('d''au plus 20 lettres en faisant suivre chacun');
writeln('d''eux de l'appui sur la touche d''entrée.');
writeln('L''arrêt se fera à STOP (en majuscules), à condition');
writeln('que vous m''ayez fourni auparavant ATTENTION.');
```

```
Compteur := 0;
repeat
     readln(MotLu);
     Compteur := Compteur +1;
until MotLu = 'ATTENTION';
repeat
     readln(MotLu);
     Compteur := Compteur +1;
until MotLu = 'STOP';
writeln('Vous m''avez donné en tout ',Compteur,' mots.');
end.
```

Cette dernière version reste à habiller en étant attentif, par des commentaires, à son lecteur potentiel:

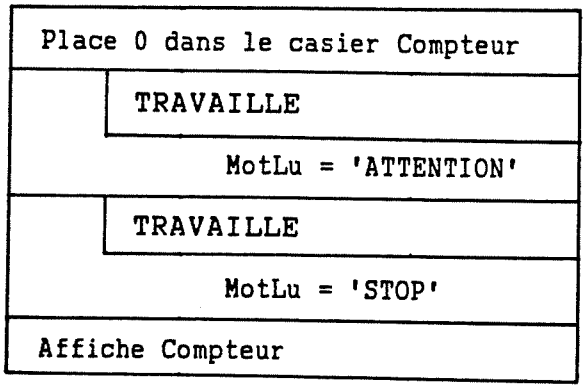
```
program ATTENTION1;

(* il fait lire des mots jusqu'à ce que STOP soit fourni à condition que ATTENTION ait été fourni auparavant
et compte les mots lus. On y utilise une double répétition *)

var MotLu
      (* qui contiendra chacun des mots lus *)
     : string[20] ;
     Compteur
     (* pour les compter *)
     : integer ;

begin
writeln('Vous allez me fournir successivement des mots');
writeln('d''au plus 20 lettres en faisant suivre chacun');
writeln('d''eux de l'appui sur la touche d''entrée.');
writeln('L''arrêt se fera à STOP (en majuscules), à condition');
writeln('que vous m''ayez fourni auparavant ATTENTION.');
Compteur := 0;
repeat
     readln(MotLu);
     Compteur := Compteur +1;
until MotLu = 'ATTENTION';
repeat
     readln(MotLu);
     Compteur := Compteur +1;
until MotLu = 'STOP';
writeln('Vous m''avez donné en tout ',Compteur,' mots.');
end.
```

Toujours avec la même structure de solution, j'avais également développé un GNS illustrant la structure d'appel de procédure. La marche à suivre "principale" comportait alors une instruction d'action complexe ("TRAVAILLE") qui se trouvait explicitée dans une marche à suivre annexe (procédure).
Les GNS correspondants étaient :



avec

Annexe : **TRAVAILLE**

Affiche 'Donnez un mot'
Lis et place dans MotLu
Place Compteur + 1 dans le casier Compteur

En voici la traduction en Pascal, illustrant la structure d'appel de procédure:

```
program ATTENTION2;

var MotLu : string[20] ;
    Compteur : integer ;

        procedure TRAVAILLE; (1)
        begin                          (2)
        write('Donnez un mot');
        readln(MotLu);            (4)
        Compteur := Compteur + 1; (4)
        end;                           (2)

begin
repeat
     TRAVAILLE;              (3)
until MotLu = 'ATTENTION';
repeat
     TRAVAILLE ;             (3)
until MotLu = 'STOP';
writeln(Compteur);
end.
```

Remarques

(1) **procedure** TRAVAILLE;

Le texte de la procédure (annexe) est annoncé par le mot réservé procedure. Il se situe entre la partie déclaration et le corps du programme principal. La procédure est désignée par un identificateur (ici TRAVAILLE) qui suit les règles habituelles.

(2) On verra que le texte d'une procédure peut comporter, après son entête, une partie déclaration; elle comporte en tout cas un corps (reprenant la traduction de la marche à suivre correspondant à l'annexe) situé (comme pour le corps du programme principal) entre les mots *begin* et *end*. Cependant, le texte de la procédure se termine par un point-virgule et non par un point final.

(3) TRAVAILLE;

L'appel de la procédure (instruction d'action trop complexe, référant à la marche à suivre annexe) se fait simplement en citant son nom (ici TRAVAILLE). Lors de la rencontre de cette instruction, l'exécutant se détourne un moment du programme principal pour suivre les instructions commandées par la procédure. Après l'exécution des actions commandées par celle-ci, il revient au programme principal, là où il l'avait laissé.

(4) Les actions commandées dans le texte de la procédure peuvent faire manipuler des variables définies dans le programme principal. Ainsi, le texte de la procédure TRAVAILLE fait mention des variables MotLu et Compteur, définies dans le programme principal. Nous reviendrons dans la suite beaucoup plus longuement sur cette possibilité.

Le programme doit, comme sa version précédente être habillé en tenant compte de son utilisateur :

```
program ATTENTION2;

var MotLu : string[20] ;
    Compteur : integer ;

        procedure TRAVAILLE;
        begin
        write('Donnez un mot : ');
        readln(MotLu);
        Compteur := Compteur + 1;
        end;

begin
writeln('Vous allez me fournir successivement des mots');
writeln('d''au plus 20 lettres en faisant suivre chacun');
writeln('d''eux de l'appui sur la touche d''entrée.');
writeln('L''arrêt se fera à STOP (en majuscules), à condition');
writeln('que vous m''ayez fourni auparavant ATTENTION.');
repeat
     TRAVAILLE;
until MotLu = 'ATTENTION';
repeat
     TRAVAILLE ;
until MotLu = 'STOP';
writeln('Vous m''avez donné en tout ',Compteur,' mots.');
end.
```

Et en tenant compte du lecteur de cette version :

```
program ATTENTION2;

(* il fait lire des mots jusqu'à ce que STOP soit fourni à condition que ATTENTION ait été fourni auparavant
et compte les mots lus. On y utilise un appel de procédure *)

var MotLu
      (* qui contiendra chacun des mots lus *)
      : string[20] ;
            Compteur
      (* pour les compter *)
      : integer ;

            procedure TRAVAILLE;
            (* elle fait lire un mot et fait incrémenter le Compteur *)
            begin
            write('Donnez un mot : ');
            readln(MotLu);
            Compteur := Compteur + 1;
            end;

begin
writeln('Vous allez me fournir successivement des mots');
writeln('d''au plus 20 lettres en faisant suivre chacun');
writeln('d''eux de l'appui sur la touche d''entrée.');
writeln('L''arrêt se fera à STOP (en majuscules), à condition');
writeln('que vous m''ayez fourni auparavant ATTENTION.');
repeat
     TRAVAILLE;
until MotLu = 'ATTENTION';
repeat
     TRAVAILLE ;
until MotLu = 'STOP';
writeln('Vous m''avez donné en tout ',Compteur,' mots.');
end.
```

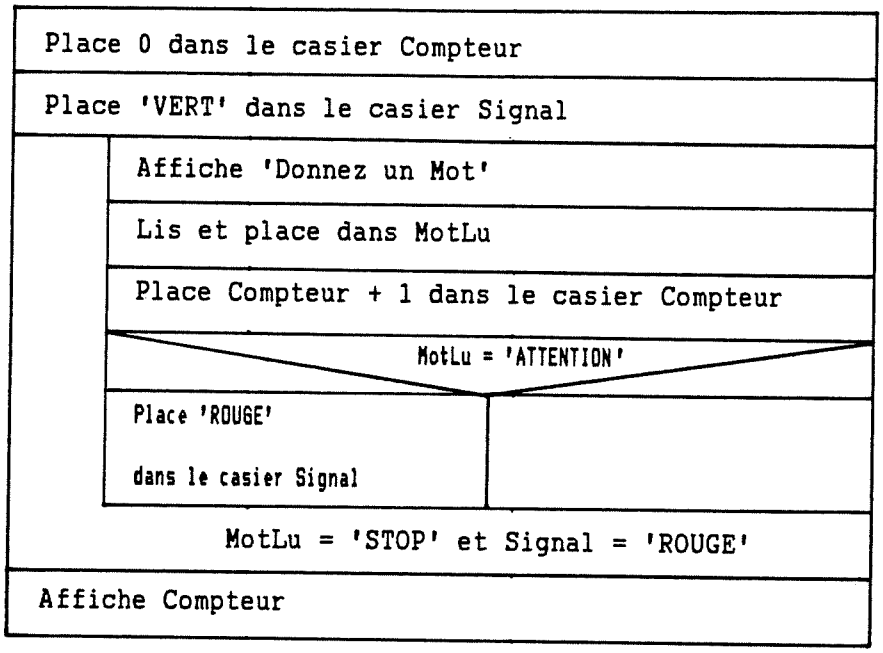
Deuxième solution

La seconde solution apportée au problème utilisait essentiellement une variable signal qui passait du "vert" au "rouge" à la réception du mot ATTENTION. Cette approche avait conduit à :

Liste des variables

- MotLu, de type chaîne d'au plus 20 caractères
- Compteur, de type entier
- Signal, de type chaîne d'au plus 5 caractères, passant de 'VERT' à 'ROUGE'

Marche à suivre



En voici la traduction en Pascal :

```
program ATTENTION3;

var MotLu : string[20];
    Compteur : integer ;
    Signal : string[5];                                    (1)

begin
Compteur := 0;
Signal := 'VERT';
repeat
     readln(MotLu);
     Compteur := succ(Compteur);                           (2)
     if MotLu = 'ATTENTION' then                           (3)
           Signal := 'ROUGE';
until (MotLu = 'STOP') and (Signal = 'ROUGE');             (4)
writeln(Compteur );
end.
```

Remarques

(1) Signal : string[5];

La variable Signal n'aura à contenir que 'ROUGE' ou 'VERT', chaînes d'au plus 5 caractères; Signal sera donc de type string[5].

(2) Compteur := succ(Compteur);

Plutôt que de désigner par "Compteur + 1" la valeur suivante prise par le casier Compteur, nous avons utilisé un outil différent de l'habituelle addition. C'est l'outil succ (successeur de) qui permet de passer d'un entier à son successeur (ce qui revient au même que de lui ajouter 1). Notons cependant que l'emploi de cet outil n'aurait aucun sens pour des informations de type real ou string. Que serait en effet le successeur de 2.23 ou de 'Bonjour'?

(3) if MotLu = 'ATTENTION' then
Signal := 'ROUGE';

La structure alternative Si ... alors ... se traduit en Pascal par if ... then ... La syntaxe en est la suivante :

if condition then instruction *unique*

ce qui nécessitera des explications lorsque le then portera sur plusieurs instructions.

(4) (MotLu = 'STOP') and (Signal = 'ROUGE')

Le connecteur logique "et" se traduit par and . Il permet de former une condition rassemblant plusieurs comparaisons. Pour que la condition entière soit vraie, il faut que chacune des comparaisons la composant le soit. De plus, les diverses comparaisons liées par and doivent être encloses dans des parenthèses.

Il resterait comme pour les deux versions précédentes à habiller ce programme d'abord en pensant à son utilisateur puis au lecteur éventuel. Je vous en laisse le soin !

----- o -----

Problème 3

Il s'agissait cette fois de faire lire des mots avec arrêt à STOP à condition que ATTENTION soit survenu juste avant.

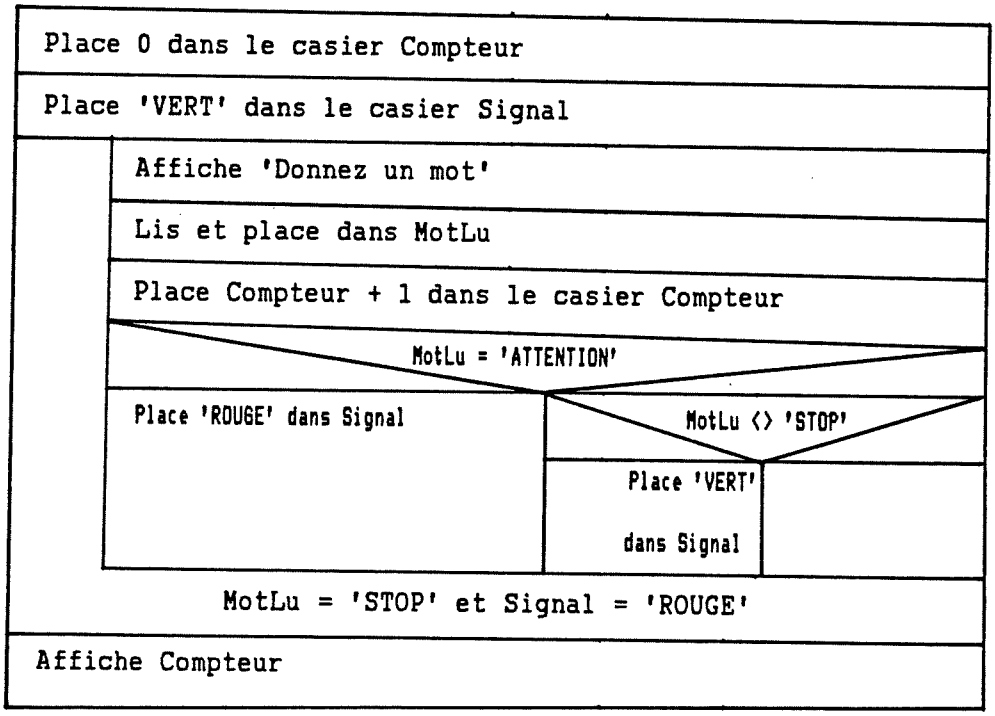
Première solution

Elle faisait à nouveau usage d'une variable Signal qui passait du 'VERT' au 'ROUGE' lorsque survenait 'ATTENTION', mais qui ne pouvait rester 'ROUGE' qui si ensuite c'était 'STOP' qui était reçu.

Liste des variables

- Motlu, de type chaîne d'au plus 20 caractères
- Compteur, de type entier
- Signal, de type chaîne d'au plus 5 caractères

Marche à suivre



Et voici la traduction Pascal de ce GNS :

```
program ATTENTIONSTOP;

var MotLu : string[20] ;
    Compteur : integer ;
    Signal : string[5];
```

```
begin
Compteur := 0;
Signal := 'VERT';
repeat
     write('Donnez un mot');
     readln(MotLu);
     Compteur := Compteur + 1;
     if MotLu = 'ATTENTION' then
          Signal := 'ROUGE'
      else                                                  (1)
          if MotLu <> 'STOP' then                  (2)
               Signal := 'VERT';
until (MotLu = 'STOP') and (Signal = 'ROUGE');
writeln(Compteur );
end.
```

(1) else

La structure alternative complète Si *condition* alors ... sinon ... se traduit par

```
if condition then ... else ...
```

La syntaxe de cette instruction Pascal est la suivante :

```
if condition then
     instruction unique
 else
     instruction unique
```

On remarquera l'absence de point-virgule avant le mot else.

(2) On peut parfaitement avoir des structures alternatives imbriquées comme ici ou l'instruction unique attendue après le else est à nouveau une alternative if ... then ... On remarquera les diverses indentations qui mettent en évidence ces structures imbriquées.

Et voici la version complètement habillée de ce programme :

```
program ATTENTIONSTOP;

(* il fait lire des mots jusqu'à ce que STOP soit fourni à condition que ATTENTION ait été fourni juste
avant et compte les mots lus. On y utilise une variable signal *)

var  MotLu : string[20] ;
                (* qui contiendra chacun des mots lus *)
     Compteur : integer ;
                (* pour les compter *)
     Signal : string[5];
                (* qui passera de 'VERT' à 'ROUGE' lorsque 'ATTENTION sera lu *)

begin
writeln('Vous allez me fournir successivement des mots');
writeln('d''au plus 20 lettres en faisant suivre chacun');
writeln('d''eux de l'appui sur la touche d''entrée.');
writeln('L''arrêt se fera à STOP (en majuscules), à condition');
writeln('que vous m''ayez fourni juste avant ATTENTION.');
```

```
Compteur := 0;
Signal := 'VERT';
repeat
     write('Donnez un mot : ');
     readln(MotLu);
     Compteur := Compteur + 1;
     if MotLu = 'ATTENTION' then
          Signal := 'ROUGE'
      else
          if MotLu <> 'STOP' then
               Signal := 'VERT';
until (MotLu = 'STOP') and (Signal = 'ROUGE');
writeln('Vous m''avez donné en tout ',Compteur,' mots.');
end.
```

Deuxième Solution

Cette seconde approche faisait retenir par l'exécutant le dernier et l'avant-dernier mots lus.

Liste des variables

- Dernier, de type chaîne d'au plus 20 caractères
- AvantDernier, de type chaîne d'au plus 20 caractères
- Compteur, de type entier

Marche à suivre

Place 0 dans le casier Compteur
Place 'BONJOUR' dans le casier Dernier
Place Dernier dans le casier AvantDernier
Affiche 'Donnez un mot'
Lis et place dans Dernier
Place Compteur + 1 dans le casier Compteur
Dernier = 'STOP' et AvantDernier = 'ATTENTION'
Affiche Compteur

Voici la traduction en Pascal de cette marche à suivre :

```
program ATTENTIONSTOP2;

var Dernier, AvantDernier : string[20];        (1)
    Compteur : integer;
```

```
begin
Compteur := 0;
Dernier := 'BONJOUR';
repeat
     AvantDernier := Dernier;
     write('Donnez un mot : ');
     readln(Dernier);
     Compteur:=Compteur+1;
until (AvantDernier='ATTENTION') and  (Dernier='STOP');
writeln(Compteur);
end.
```

(1) `var Dernier, AvantDernier : string[20];`

Lorsque plusieurs variables du même type doivent être définies, il est loisible d'en dresser la liste à condition de séparer leurs noms par la virgule. Comme toujours, cette liste est suivie de deux points et du type des variables de la liste.

On aurait cependant pu écrire également

```
var  Dernier : string[20];
     AvantDernier : string[20];
     Compteur : integer;
```

Il reste une fois de plus à habiller le programme. Cet habillage sera en grande partie identique à ce qu'il est pour le précédent... et je vous en laisse donc le soin.

----- o -----

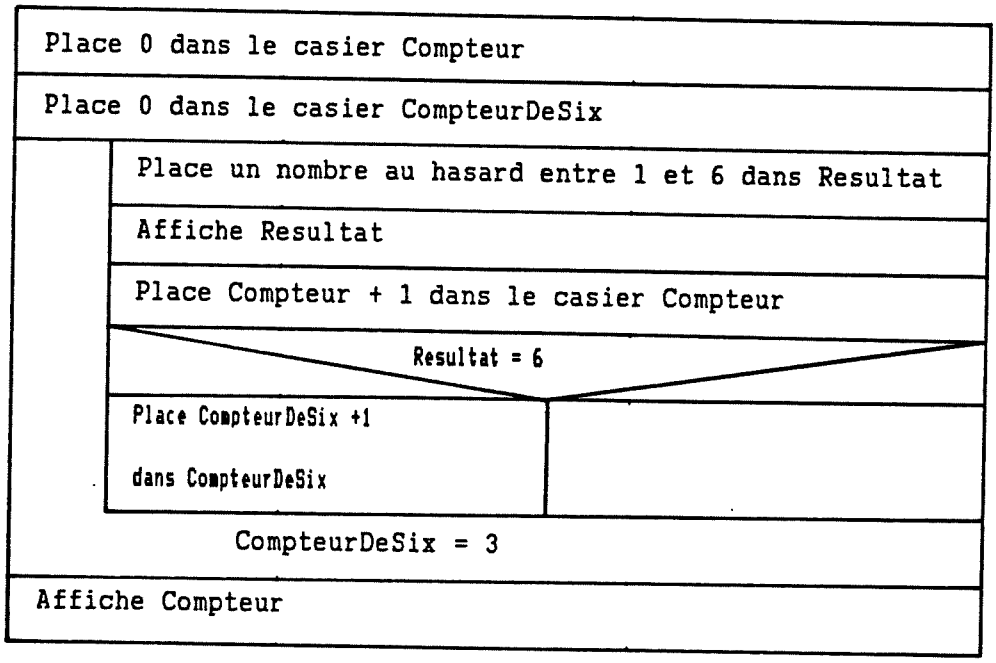
Problème 4

Nous avions rédigé un GNS qui faisait simuler des jets successifs d'un dé jusqu'à l'obtention du troisième six :

Liste des variables

Resultat, de type entier, qui contiendra un instant chaque résultat du tirage au hasard
Compteur, de type entier, pour compter les lancers; il s'augmentera de 1 à chaque lancer
CompteurDeSix, de type entier, pour compter les six qui se présenteront.

Marche à suivre



La traduction en est à nouveau immédiate : le seul petit problème est d'indiquer comment, en Pascal (dans l'implémentation Turbo), on peut parler d'un nombre (entier) au hasard entre 1 et 6.

```
program JETDEDES;

var Resultat, Compteur, CompteurDeSix : integer;      (1)

begin
Compteur := 0;
CompteurDeSix:=0;
repeat
     Resultat := random(6)+1;       (2)
     write(Resultat);
     Compteur:=succ(Compteur);
     if Resultat=6 then
          CompteurDeSix:=succ(CompteurDeSix);
until CompteurDeSix=3;
writeln(Compteur);
end.
```

(1) var Resultat, Compteur, CompteurDeSix : integer;

Nous avons à nouveau dressé une seule liste de toutes les variables entières.

(2) Resultat := random(6)+1;

L'outil qui permet à l'éxécutant de disposer d'un nombre entier tiré au hasard s'appelle random. L'exécutant y place un nombre (ici 6) à l'entrée et il y reprend un nombre entier compris entre 0 (inclus) et le nombre entré (exclus) (ici donc entre 0 et 5). Il ne reste plus alors qu'à ajouter 1 à l'information ainsi obtenue pour disposer à coup sûr d'un nombre entre 1 et 6.

L'habillage de ce programme va (pour une fois) nous amener à découvrir quelques possibilités supplémentaires du langage Pascal (dans son implémentation Turbo).

```
program JETDEDES;
(* Il fait simuler des jets successifs d'un dé jusqu'à obtention d'un troisième 6 *)

var Resultat,
      (* qui contiendra les résultats du "lancer" *)
      Compteur,
      (* pour compter le nombre de lancers *)
      CompteurDeSix
      (* pour compter les 6 obtenus *)
       : integer;


begin
(* effacement de l'écran *)
clrscr;                    (1)

writeln('Je vais lancer pour vous un dé en arrêtant');
writeln('au troisième 6 obtenu.');
(* demande d'une pause *)
delay(5000);               (2)

clrscr;                    (1)
(* placement du curseur à la 2ème ligne, 36ème colonne *)
gotoxy(36,2);              (3)

write('Résultats');        (4)
(* placement du curseur à la 3ème ligne, 36ème colonne *)
gotoxy(36,3);              (5)

writeln('=========');
(* passage d'une ligne blanche *)
writeln;                   (6)
```

```
Compteur := 0;
CompteurDeSix:=0;
repeat
     Resultat := random(6)+1;
     write(Resultat);
     Compteur:=succ(Compteur);
     if Resultat=6 then
          CompteurDeSix:=succ(CompteurDeSix);
until CompteurDeSix=3;
writeln;
writeln('Nombre total de jets : ',Compteur);
end.
```

(1) *clrscr;*

Cette instruction exige que l'exécutant efface complètement l'écran. Il s'agit d'un raccourci pour les mots "clear screen" qui signifie de fait "efface l'écran". C'est une sage précaution, en début de programme de faire procéder à cet effacement.

On pourrait sans doute voir dans l'instruction clrscr une instruction d'action élémentaire ne correspondant à aucune des trois actions élémentaires dont est capable l'exécutant. On pourrait aussi y voir un racourci pour "Affiche ... un écran vide" et l'assimiler à un cas (très) particulier de l'instruction d'affichage.

(2) *delay(5000);*

L'instruction delay(...) fait temporiser l'exécutant pendant un moment proportionnel à la taille du nombre entier inscrit entre les parenthèses. (Ce temps dépend de l'ordinateur utilisé, mais il doit en général être de quelques milliers pour que la temporisation soit "visible"). Pendant ce moment, l'exécutant ne poursuit pas la marche à suivre et tout reste donc inchangé.

Pourquoi donc "stopper" ainsi l'exécutant pendant un temps plus ou moins long ? Il faut remarquer qu'ici, sans cette instruction, le morceau de programme concerné serait

```
clrscr;
writeln('Je vais lancer pour vous un dé en arrêtant');
writeln('au troisième 6 obtenu.');
(* demande d'une pause *)
clrscr;
```

Dès lors, après avoir effacé l'écran (*clrscr*), l'exécutant y afficherait les deux lignes de message exigées, puis immédiatement après effacerait de nouveau, faisant aussitôt disparaître les deux lignes qu'il venait d'afficher. Et comme notre exécutant est (heureusement en général, malheureusement ici) fort rapide l'utilisateur n'aurait absolument pas eu le temps de lire notre message! Nous commandons donc à l'exécutant de rester un moment "à ne rien faire", avant d'effacer, moment pendant lequel l'utilisateur aura le loisir de lire les deux lignes de présentation de l'objectif du programme ... puis les choses reprendront leur cours.

Il s'agit réellement cette fois d'une instruction d'action élémentaire nouvelle. Elle ne figurait pas au répertoire de l'exécutant tel qu'il avait été décrit dans un chapitre précédent. J'ai donc menti (par omission !). Deux remarques à ce propos:

d'abord cette action nouvelle ne change pas énormément le portrait de l'exécutant gestionnaire de casiers, avec sa porte, sa fenêtre et les trois actions dont il est capable; ensuite, cette omission (volontaire) est la première d'une longue série que nous digérerons petit à petit.
Il s'agit là de détails: ce n'est pas la connaissance de ceux-ci qui fait la différence entre les "bons programmeurs" et les "autres". Mais autant savoir ...

(3) (et (5)) *gotoxy(36,2);*

Et voici à nouveau une instruction de présentation des informations affichées à l'écran. Il s'agit ici de positionner le curseur à un endroit déterminé de ce dernier. Le curseur, c'est ce petit rectangle (ou carré, ou tiret ...) que vous voyez à tout moment à l'écran. Le texte qui sera affiché à l'écran, d'où qu'il vienne (affiché par l'exécutant ou fourni par l'utilisateur à travers son clavier vient toujours prendre place à partir de la position du curseur. Le fait d'envoyer le curseur à un endroit précis, préalablement à un affichage de texte, permet donc une présentation plus attrayante des messages ou des résultats.

L'écran est (en général) constitué de 24 lignes de 80 caractères. On peut à l'aide de gotoxy(.....) envoyer le curseur à n'importe lequel de ces emplacements de l'écran. Les arguments de cette instruction (ce sont les deux nombres écrits dans les parenthèses) doivent être des informations entières (écrites comme des constantes, des variables ou des expressions). Le premier argument indique la colonne où se déplacera le curseur (elle détermine donc son déplacement horizontal); le second précise la ligne où il sera envoyé (déplacement vertical).

Ainsi,

gotoxy(36,2)

enverra le curseur à la 36ème colonne, 2ème ligne (en haut et à peu près au milieu de l'écran).

(4) *write('Résultats');*

Le titre dont write commande l'affichage viendra donc s'inscrire (à cause de la position prise par le curseur après l'instruction gotoxy qui précédait) sur la 2ème ligne (en haut de l'écran), à partir de la 36ème position. Ce titre sera ainsi (à peu près centré).

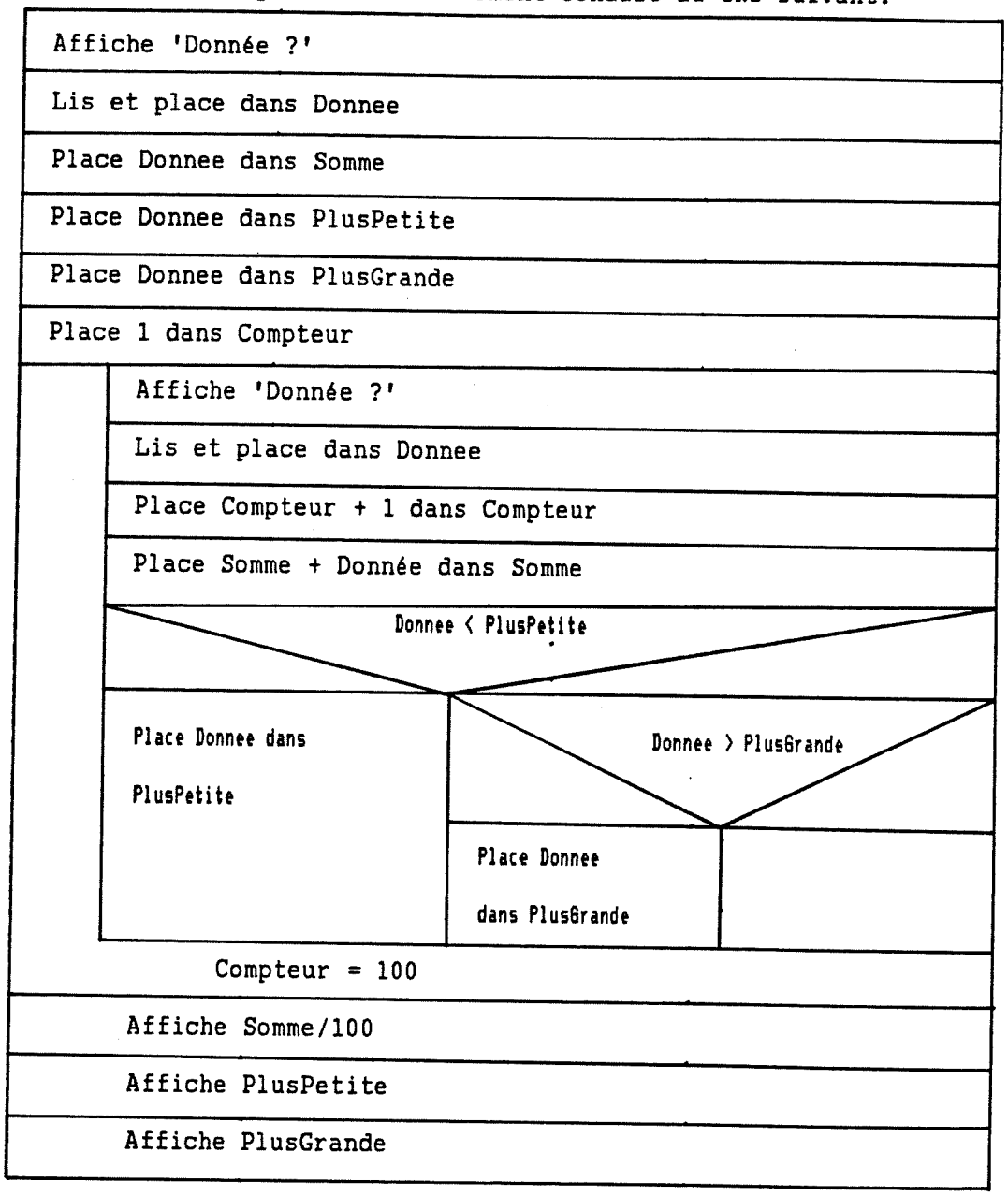
(6) *writeln;*

L'instruction writeln employée seule (sans argument) commande un saut à la ligne.

----- o -----

Problème 5

Il s'agissait cette fois de faire lire une série de 100 nombres réels et de fournir ensuite la plus grande donnée, la plus petite et leur moyenne. Notre analyse avait finalement conduit au GNS suivant:



Voici le programme Pascal correspondant :

```
program MOYENNE1;

var  Donnee, Somme, PlusPetite, PlusGrande : real;          (1)
     Compteur : integer;
```

```
begin
write('Donnée : ');
readln(Donnee);
Somme:=Donnee;
PlusPetite:=Donnee;
PlusGrande:=Donnee;
Compteur:=1;
repeat
     write('Donnée : ');
     readln(Donnee);
     Compteur:=succ(Compteur);
     Somme:=Somme+Donnee;
     if Donnee<PlusPetite then
          PlusPetite:=Donnee
      else
          if  Donnee>PlusGrande then
               PlusGrande:=Donnee;
until Compteur=100;
writeln(Somme/100);                    (2)
writeln(PlusPetite);
writeln(PlusGrande);
end.
```

(1) `var Donnee, Somme, PlusPetite, PlusGrande : real;`

Le type réel se traduit `real`. Une variable de ce type pourra donc contenir un nombre réel. Le langage Pascal permet cependant, outre l'affectation d'une information réelle à une variable réelle, l'affectation à celle-ci d'une information entière. Ainsi, on pourrait écrire

```
Donnée := 13.567
```

mais aussi

```
Donnée:= 13
```

ou

```
Donnee:=Compteur
```

Donnee étant une variable réelle et Compteur une variable entière.

L'inverse (affectation d'une information de type réel à une variable entière) n'est évidemment pas permis.

(2) Somme/100

Nous trouvons ici, pour la première fois, l'opération de division. L'outil correspondant figure bien entendu sur la table de travail de l'exécutant. Les deux informations à fournir doivent être de type entier ou réel; le résultat lui est toujours de type réel.

Ainsi

```
4/2 est de type réel
```

comme

```
Somme/1000
```

même si, dans ce dernier cas, la division "tombe juste".

C'est une règle (assez) générale qu'on puisse ainsi lorsqu'on commande une opération (+,-,*,/) mélanger des arguments entiers et réels. Si les deux nombres sont de type entier, le résultat l'est lui aussi (sauf pour /), sinon le résultat est de type réel.

Comme d'habitude, il reste à habiller le programme ainsi obtenu :

```
program MOYENNE1;
(* il fait lire une série de 100 données réelles et fait afficher la plus petite, la plus grande et leur
moyenne *)


var Donnee,
      (* qui contiendra successivement les diverses données fournies *)
      Somme,
      (* qui contiendra la somme des données *)
      PlusPetite,
      (* qui finira par contenir la plus petite donnée *)
      PlusGrande
      (*   qui contiendra la plus grande donnée *)
      : real;
      Compteur
      (* pour compter les 100 données lues *)
      : integer;

begin
writeln('Vous allez me fournir 100 données réelles et');
writeln('je vous donnerai ensuite la plus petite, la plus grande');
writeln('et leur moyenne.');
writeln;
write('Votre première donnée : ');
readln(Donnée);
Somme:=Donnee;
PlusPetite:=Donnee;
PlusGrande:=Donnee;
Compteur:=1;
repeat
     write('Donnée numéro ',Compteur+1,' : '); (1)
     readln(Donnee);
     Compteur:=succ(Compteur);
     Somme:=Somme+Donnee;
     if Donnee<PlusPetite then
          PlusPetite:=Donnee
      else
          if  Donnee>PlusGrande then
                PlusGrande:=Donnee;
until Compteur=100;
clrscr; (* effacement de l'écran *)
writeln('La moyenne des données est : ',Somme/100);
writeln('La plus petite est : ',PlusPetite);
writeln('et la plus grande : ',PlusGrande);
end.
```

(1) `write('Donnée numéro ',Compteur+1,' : ');`

Au moment d'inviter l'utilisateur à fournir une donnée, je lui rappelle (ou, plutôt fais rappeler) la position de celle-ci. L'affichage résultant sera du type

```
Votre première donnée : 23.45
Donnée numéro 2 : -34.67
Donnée numéro 3 : 23
      ...         ...
```

les quantités soulignées étant les nombres fournis par l'utilisateur. Il est important de bien voir pourquoi je commande

```
write('Donnée numéro ',Compteur+1,' : ')
```

et pas

```
write('Donnée numéro ',Compteur,' : ')
```

----- o -----

EXERCICES

I. Traduisez en Pascal et "habillez" les programmes obtenus pour chacun des GNS développés dans les exercices proposés au chapitre précédent.

II. A et B désignant deux quantités entières (avec A<B), par quelle expression désigneriez-vous en Pascal un nombre entier aléatoire compris entre A et B (A et B inclus)?

III. Pourquoi ne redouble-t-on pas les apostrophes qui figurent dans les commentaires ?

----- o -----

SYNTHESE SUR LE LANGAGE

I. STRUCTURE GENERALE D'UN PROGRAMME PASCAL

1. Entete du programme.

La première ligne comporte nécessairement l'indication

program *NomChoisi* ;

Les mots réservés

Le mot program est ce qu'un appelle un mot réservé du langage Pascal. Nous en connaissons beaucoup d'autres : var, repeat, readln, until, then, ...Nous avons pris l'habitude de les souligner dans les exemples précédents. Ces mots réservés ne peuvent évidemment être utilisés librement pour désigner le programme, les variables, procédures ...

Ces mots réservés doivent être écrits tels quels, sans y faire figurer d'espace, de tirets, de passage à la ligne, ... Ils doivent être suivis d'au moins un caractère permettant de voir qu'ils sont bien terminés. Ces caractères "terminateurs" peuvent être l'espace, la virgule, le point-virgule,le passage à la ligne, (, +, /, -, ... ou tout autre symbole de "ponctuation" admis par le langage.

Ainsi,

programattention
varcompteur

sont incorrects puisqu'ils ne permettent pas d'identifier les mots réservés.

Lorsque nous en serons à la toute dernière étape de la démarche, face à l'ordinateur et en train de lui fournir les textes des programmes que nous aurons conçus, ces textes devront être dépouillés de tous les ornements (soulignés, caractères gras, italiques, ...) qui figurent dans les textes repris ici. Ainsi, les mots réservés ne pourront pas être soulignés. (Cela ne sera d'ailleurs pas possible).

Les identificateurs

NomChoisi est le nom que l'on désire donner au programme.

- *NomChoisi* est ce que nous appellerons par la suite un identificateur : c'est, nous venons de le voir, un identificateur qui désignera le programme, mais aussi les variables (casiers) et les procédures utilisées.

Un identificateur est une succession de lettres et de chiffres à l'exclusion de tout autre symbole sauf le caractère de soulignement _), commençant obligatoirement par une lettre. Les lettres peuvent être écrites en majuscules ou en minuscules mais Pascal ne fait pas de différence : pour lui A et a, c'est la même lettre (sauf lorsqu'on lui parlera de constantes chaîne de caractère, ce qui n'a rien à voir avec les identificateurs). Notez cependant que, pour les identificateurs, les lettres accentuées (é, è, ê, à,...) ne sont pas acceptées.

- Il va sans dire que l'identificateur du programme ne peut être réemployé tel quel pour identifier dans la suite des variables, des procédures, etc... . De plus, nous l'avons dit ci-dessus, les mots réservés du langage (program, var, ...)ne peuvent servir comme identificateurs. Ces mots réservés peuvent cependant être inclus dans des identificateurs valables : par exemple, *programme*, *variable*, *difference*, ... sont permis.

- Exemples :

```
FACTORIELLE

EQUATIONDUSECONDDEGRE

EQUATIONDUPREMIERDEGRE

EquationDuPremierDegre

A123
```

sont des identificateurs valables.

Par contre, ne conviennent pas :

```
1A3              (Commence par un chiffre)

Program          (Mot-clé réservé)

T**              (Symboles * inacceptables)

Début            (Comprend une lettre accentuée)

Un Grand         (Comporte un espace blanc)

Age
duCapitaine      (Comporte un passage à la ligne)
```

- C'est une excellente habitude de se servir d'identificateurs qui "disent quelque chose". Cela permet souvent une relecture plus aisée des programmes.

- Par ailleurs, puisque le compilateur ne fait aucune différence entre lettres majuscules et minuscules, nous emploierons les unes et les autres, au mieux, de façon à rendre les programmes le plus lisible possible.

- Enfin, comme pour les mots réservés, un identificateur doit être suivi d'au moins un symbole permettant de voir qu'il est terminé : espace, passage à la ligne, point-virgule, virgule, :=, +, (, ...

2. Partie déclarative.

Après l'entête vient ensuite la **partie déclarative** qui reprend, entre autres, la liste des variables qui vont être employées dans le programme. Nous verrons par la suite que c'est ici aussi que seront déclarés les constantes, types, etc... .

Cette déclaration prend, par exemple, la forme:

```
var Nombre,X1,X2 : integer;
    Toto, Tutu, Turlututu : real;
    DisMoiOuiOuNon, Signe : string[80];
```

le mot var, annonçant la liste des variables de divers types n'étant écrit qu'une fois.

Nous retiendrons, pour l'instant, trois types de variables :

entières (integer)
réelles (real)
chaînes de caractères (string[ ])

correspondant aux trois types d'informations que nous avons vu manipuler jusqu'à présent par l'exécutant-ordinateur.
Rappelons qu'il est indispensable, en ce qui concerne les variables string, d'indiquer, entre crochets, la taille maximale des chaînes de caractères qui pourront y résider. Cette taille maximale doit être comprise entre 1 et 255.

> Nous découvrirons dans la suite d'autres types d'informations manipulées par l'exécutant ordinateur, donc d'autres types de variables susceptibles de les accueillir. Les trois types répertoriés jusqu'ici sont amplement suffisants pour débuter ...

3. Le corps du programme.

Enfin, le **corps du programme** comporte les instructions qui seront exécutées. Cette partie prend la forme :

begin

Une série d'instructions séparées par des ;

end.

Le dernier symbole du programme étant toujours un point.

Présentation

Les blancs, les indentations (quand une partie du texte est décalée), les passages à la ligne n'ont (en général) aucune importance au point de vie syntaxique (compte tenu du fait qu'ils ne peuvent figurer dans les mots réservés et les identificateurs). Nous les utiliserons cependant pour faire clairement apparaître la structure du programme.

Ainsi, le programme

```
program ArretA0;

var NombreFourni : real ;
    Compteur : integer ;

begin
Compteur := 0;
repeat
     readln(NombreFourni);
     Compteur := Compteur +1;
until NombreFourni = 0;
writeln (Compteur);
end .
```

pourrait aussi s'écrire

```
program ArretA0;var NombreFourni : real ;Compteur : integer ;begin Compteur
:= 0;repeat readln(NombreFourni);Compteur := Compteur +1;until NombreFourni
= 0;writeln (Compteur);end .
```

ou encore

```
program
ArretA0
;
var
NombreFourni
:
 real
;
Compteur
 : integer ;
begin
Compteur := 0;
```

etc, etc ...

Inutile de souligner que la première version est plus lisible que les deux suivantes.

Les instructions en Pascal

Ainsi que nous le découvrirons ci-dessous, il ne faut pas confondre ce que nous appellerons "instruction" en Pascal et le concept d'instruction d'action élémentaire que nous connaissons.

Bien entendu, les trois instructions d'actions élémentaires donneront naissance aux trois instructions Pascal d'affectation (:=), de lecture (readln) et d'affichage (write ou writeln). Mais la traduction en Pascal des structures de contrôle donnera lieu à ce que nous appellerons aussi instruction en Pascal (repeat ... until ..., while ... do ..., if... then ...else ..., etc...).

Nous parlerons donc de l'instruction repeat ... until ..., de l'instruction if... then ..., etc

Les "terminateurs" d'instruction Pascal

C'est, on l'a vu, le symbole point-virgule qui permet d'indiquer qu'une instruction est terminée (et non le passage à la ligne).
Il faut noter cependant que ***begin*** n'est pas une instruction, repeat non plus. C'est en réalité ***begin*** ... ***end*** qui constitue une instruction, comme aussi repeat ... until ... dans son ensemble.

On peut donner l'impression à l'exécutant que des instructions successives constituent en fait une instruction unique. Il suffit de faire précéder la première instruction de la série par ***begin*** et de faire suivre la dernière de ***end***. Cette globalisation est d'ailleurs parfois obligatoire (comme après le then ou le else); nous y reviendrons.

----- o -----

II. QUELQUES INSTRUCTIONS PASCAL FONDAMENTALES

Nous passerons d'abord en revue les instructions Pascal correspondant aux trois instructions d'actions élémentaires "comprises" par l'exécutant.

1. L'instruction d'affectation :=

Elle correspond à l'instruction notée dans les GNS par

Place *telle information* dans *tel casier*

La traduction va malheureusement inverser l'ordre dans lequel se présentaient l'information et le casier à remplir puisque l'instruction d'affectation va se traduire :

nom du casier := information à y placer

Elle permet de placer dans la variable (casier) située à gauche du signe := l'information décrite à droite. Rappelons que cette information s'écrira comme une constante, une variable (dont on désigne alors le contenu), ou encore une expression (résultat d'une manipulation mettant en jeu l'un ou l'autre outil de traitement des informations).

Exemples :

```
Debut := 15

Rho := Epsilon

Indice := Indice + 1

VALEUR := ACC * (TYU-TRE)/BIBI
```

Il faut bien sûr veiller, lors de l'emploi de l'affectation, à la compatibilité du type de la variable à remplir (tel qu'il a été précisé dans la partie déclaration du programme) et de l'information qu'on demande

d'y placer. Par exemple, il est impossible d'affecter à une variable entière une information réelle ou chaîne de caractères (qu'elle prenne la forme d'une constante, d'une variable ou d'une expression). La seule exception permise est l'affectation d'une information entière à une variable réelle.

2. Les instructions de lecture READ et READLN

Elles permettent de faire lire, à partir du clavier, les informations (données) fournies par l'utilisateur. Ces données sont placées dans les casiers indiqués. read et readln traduisent donc l'instruction notée dans les GNS par

Lis et place dans *tel casier*

la traduction en Pascal devenant

read(*nom du casier à remplir*)

ou encore

readln(*nom du casier à remplir*)

la différence entre ces deux formes étant (pour l'instant et dans l'implémentation Turbo) sans importance.

Il faut cependant souligner une possibilité (qui sera rarement employée) et que nous n'avions pas mentionnée dans les GNS : il est permis de faire lire par une seule instruction de lecture plusieurs informations qui devront bien entendu prendre place dans des variables différentes. Il suffit de faire suivre l'instruction read ou readln de la liste des variables à remplir par les diverses informations qui seront fournies par l'utilisateur. Les diverses variables citées seront séparées par des virgules.

Par exemple

read(Nom, Age, Profession)

readln(X, Somme)

Il est, en général préférable de réserver ces lectures multiples aux cas d'informations numériques.

L'instruction de lecture provoque un arrêt du déroulement des actions de l'exécutant qui attend que la (les) donnée(s) adéquate(s) soi(en)t tapée(s) au clavier. L'utilisateur signalera la fin de l'information fournie par l'appui sur la touche d'entrée ⟨-┘ aussi appelée RETURN ou ENTER.

3. Les instructions d'écriture WRITE et WRITELN

Ce sont les instructions Pascal correspondant à l'instruction d'action élémentaire

Affiche *telles informations*

Elles font apparaître à l'écran les informations indiquées. Elles se traduisent :

write(*informations à afficher*)

ou encore

writeln(*informations à afficher*)

Par exemple :

```
write('Resultat : ',Nombre)

writeln(X,Y,Somme)

writeln(2*X,' : ',15)
```

On le voit, les informations à afficher prennent (comme toujours) la forme soit de constantes ('Resultat : ',15), soit de variables (Nombre, X, Y, Somme), soit encore d'expressions (2*X).

Les diverses informations dont on demande l'affichage doivent être séparées par des virgules. Elles seront affichées les unes à la suite des autres, collées l'une à l'autre.

L'instruction writeln provoque, APRES l'impression des informations concernées un saut à la ligne. Les informations qui apparaitront donc ensuite à l'écran, qu'elles y soient à cause d'un nouvel affichage ou à cause d'une lecture (les informations frappées par l'utilisateur apparaissant (évidemment !) à l'écran au fur et à mesure qu'il les frappe), seront situées à la ligne suivante.
L'instruction writeln peut aussi être utilisée seule (sans les parenthèses) pour effectuer un passage à la ligne.

Après write, il n'y a pas de saut à la ligne.

Il est donc possible par write et writeln de faire apparaître à l'écran un "message". Il s'agit simplement de commander l'affichage d'une constante de type chaîne de caractères : il faut, rappelons-le, que celle-ci apparaisse entre deux symboles apostrophe. Par exemple, si la variable X contient la valeur 17, l'instruction

```
writeln('La valeur de X est ',X,' unités')
```

affichera à l'écran

```
La valeur de X est 17 unités
```

suivi d'un saut à la ligne.

Il importe d'être attentif, lors de l'affichage de plusieurs informations au fait qu'elles apparaîtront collées l'une à l'autre. Ainsi, en reprenant l'exemple ci-dessus, si l'on avait exigé

```
writeln('La valeur de X est',X,'unités')
```

l'affichage aurait été

```
La valeur de X est17unités
```

Il nous faut à présent passer en revue les instructions Pascal correspondant aux structures de contrôle.

4. L'instruction alternative IF...THEN...ELSE...

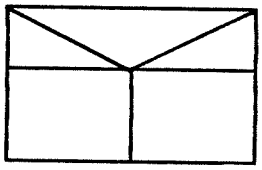
C'est celle qui traduira les mots

SI ... ALORS ... SINON ...

ou encore (en l'absence de SINON)

SI ...ALORS ...

représenté dans les GNS par le graphisme



Elle prend généralement la forme :

IF condition

THEN instruction unique

ELSE instruction unique

et, en l'absence de sinon,

IF condition

THEN instruction unique

Nous trouvons ici l'un des cas où la syntaxe de Pascal commande impérieusement de pouvoir amalgamer plusieurs instructions en une instruction unique à l'aide de *begin* et *end*.

Par exemple :

```
if A = B-C then
  begin
  C := B+C;
  A := 2*C;
  end
 else
  A := 3*A;
```

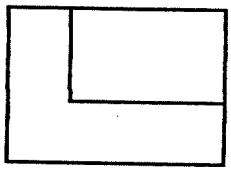
Ajoutons encore que la partie ELSE ... peut être absente. L'ensemble ifF...thenN...else... forme une instruction unique pour Pascal; on ne pourra donc JAMAIS trouver de ; juste avant le mot else (c'est pourtant une erreur très classique).

5. L'instruction de répétition REPEAT...UNTIL...

C'est elle qui traduira la structure

REPETER ... JUSQU'A CE QUE ...

représentée par le graphisme



Elle prend la forme :

```
REPEAT
  série d'instructions
UNTIL  condition
```

Les instructions situées entre le repeat et le until seront répétées jusqu'a ce que la condition devienne vraie. Dans tous les cas cette série d'instructions sera exécutée au moins une fois.

Exemple :

```
repeat
  I:=I+1;
  Somme := Somme + Ajout;
  Reste := Reste - 1;
until I>Max
```

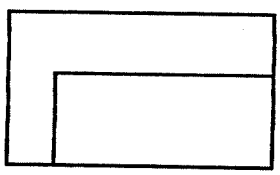
Nous prendrons l'habitude d'indenter le texte de nos programmes comme dans l'exemple ci-dessus. On peut ainsi voir d'un coup d'oeil où commence et où se termine la série des instructions qui seront répétées.

6. L'instruction de répétition WHILE...DO...

Nous n'avons pas encore eu l'occasion de l'utiliser. Elle traduit la structure

TANT QUE ... FAIRE ...

représentée par le graphisme



Elle prend la forme :

```
WHILE condition DO
  instruction unique
```

Voilà encore un cas où l'on est forcé de rassembler plusieurs instructions, que l'on veut voir exécuter à la suite de do, en une seule à l'aide de *begin* et *end*.

Par exemple :

```
while (Total <> TT) or (S=U) do
  begin
  TT := TT+1;
  Produit := Produit * (Rest + 4.1);
  end;
```

Contrairement à l'instruction repeat, l'instruction while peut voir les instructions qu'elle contrôle ne pas être exécutées une seule fois si la condition sur laquelle porte le while a la valeur FAUX dès le début.

7. L'appel de procédure.

Pascal permet de faire figurer dans le texte du programme des actions complexes qui sont explicitées dans des procédures annexes. Il s'agit là de la possibilité d'appel de procédure.

Il faut bien distinguer à ce propos deux choses :

- l'action complexe commandée dans le texte du programme (appel de la procédure)
- l'explicitation de cette action (texte de la procédure)

> Je n'essayerai pas pour l'instant d'être complet à ce propos. Beaucoup de concepts importants viendront dans la suite enrichir cette structure (variables globales ou locales, paramètres, arboresence des divers niveaux de procédures, ...).

Appel d'une procédure

L'appel d'une procédure consiste à simplement citer l'identificateur par laquelle on la désigne. Cet identificateur suit les règles syntaxiques habituelles.

Texte d'une procédure

Ce texte doit figurer au sein du programme lui-même, entre la partie déclarative et le corps du programme. L'identificateur de la procédure (ce qui dans le texte du programme sert à l'appeler) doit être précédé du mot réservé procedure et suivi d'un point-virgule : le tout constitue l'entête de la procédure. Cet entête peut être suivi d'une partie déclarative (nous y reviendrons). Vient enfin le corps de la procédure, enclos entre les mots *begin* et *end*, suivi d'un point-virgule.

Le texte d'une procédure est donc rigoureusement identique à celui d'un programme sauf au niveau de l'entête où le mot procedure remplace le mot program et tout à la fin où le point-virgule remplace le point final du programme.

Variables globales et variables locales

Un mot encore, très incomplet, sur le fait qu'on peut donc définir des variables à l'intérieur du texte d'une procédure : ces variables, locales à cette procédure, peuvent évidemment être utilisées dans le texte de celle-ci, mais ne peuvent l'être dans le corps du programme principal ou d'autres procédures.

Les variables définies au niveau du programme principal, elles, peuvent être utilisées partout, tant dans le corps du programme que dans celui des procédures qui y sont nichées.

Nous avons mis en évidence à côté des instructions Pascal correspondant aux trois instructions d'actions élémentaires et aux structures de contrôle quelques instructions auxiliaires :

8. Instructions diverses

1) L'instruction delay( ) permet d'interrompre pendant un moment l'exécution de la suite du programme. L'information à faire figurer entre les parenthèses doit être de type entier.
2) L'instruction gotoxy(,) permet de placer le curseur en un endroit précis de l'écran. Les deux quantités indiquées entre parenthèses doivent être de type entier: elles précisent respectivement la position horizontale (entre 1 et 80) et la position verticale (entre 1 et 24).
   Ainsi gotoxy(1,1) envoit le curseur dans le coin supérieur gauche et gotoxy(80,24) dans le coin inférieur droit de l'écran.
3) L'instruction clrscr fait effacer l'écran et amène le curseur dans le coin supérieur gauche (en position 1, 1).

Nous avons signalé qu'à côté des instructions d'actions élémentaires et des structures de contrôle, les marches à suivre comporteraient également des commentaires. Il nous reste à dire comment ceux-ci peuvent s'insérer dans le texte d'un programme Pascal.

9. Les commentaires

Les programmes, surtout s'ils sont longs, ne sont pas toujours aisés à relire. Le langage Pascal nous autorise (et le bon sens nous le conseille vivement) à insérer dans le texte des commentaires qui ne seront pas pris en compte par le compilateur. Ces commentaires doivent être écrit entre les symboles (* et *) ou bien { et }.

Le texte écrit entre les deux symboles (* et *) n'est soumis à aucune contrainte syntaxique

Exemple :

```
(* Ceci est un commentaire. J'y écris ce que
                je veux, comme je veux. *)
```

III. LES OUTILS DISPONIBLES

Nous avons rencontré un certain nombre d'outils de traitement des informations à l'occasion de la solution des problèmes.

1. Les opérations arithmétiques

Certains outils s'appliquent à des nombres et fournissent comme résultat un nombre. Ce sont les opérations habituelles :

+ pour l'addition
- pour la soustraction
* pour la multiplication
/ pour la division

Lorsque les opérandes (les nombres sur lesquels porte l'opération) sont de type entier, le résultat est aussi de type entier, sauf pour la division où le résultat est toujours de type réel.
Lorsqu'une des opérandes ou les deux sont de type réel, le résultat est de type réel.

A ces opérations, il faut en ajouter deux qui concernent spécifiquement le type entier et que nous n'avons pas eu l'occasion d'employer jusqu'à présent :

DIV pour la division entière
MOD pour le reste d'une division entière.

div désigne le quotient de la division entière. Ainsi

7 div 3 vaut 2 (reste 1)
15 div 4 vaut 3 (reste 3)

mod désigne le reste de la division entière:

7 mod 3 vaut 1
15 mod 4 vaut 3

Les opérandes doivent être de type entier et le résultat l'est également.

La priorité des opérateurs est celle habituelle en mathématique :

*, /, DIV et MOD ont la priorité 1
+ et - ont la priorité 2

On pourrait ajouter que les parenthèses ( ) et certaine fonctions que nous verrons par la suite ont la plus grande priorité.
Signalons aussi qu'il vaut mieux quelques parenthèses superflues que des expressions illisibles.

Pour une même priorité, les opérations sont effectuées de gauche à droite. Par exemple :

A+B*C est équivalent à A + (B * C)

A*B/C*D est équivalent à (A * B * D) / C

C-D+E/K est équivalent à (C - D) + (E / K)

Signalons enfin qu'il n'existe pas d'opération d'élévation à la puissance (exponentiation). Nous devrons donc, dans un premier temps, évaluer les puissances en les transformant en multiplications.
Par exemple :

A^4 se calcule par A*A*A*A

2. Les fonctions arithmétiques

Nous n'en avons découvert que deux jusqu'à présent:

- la fonction succ( ) (successeur de) qui s'applique à une information entière et fournit un résultat entier. Elle a le même effet que d'ajouter 1. Je peux cependant déjà signaler qu'il existera aussi une fonction pred( ) qui, appliquée à un nombre entier, fournira le prédécesseur de cet entier. Ces fonctions n'ont pas de sens dans le cas d'un argument réel ou chaîne de caractères.

- la fonction random( ) qui désigne un entier aléatoire compris entre 0 (inclus) et l'argument (exclus).
  Ainsi

  random(5) fournit au hasard l'un des entiers 0,1,2,3 ou 4.

3. Les expressions "booléennes"

Ce sont celles que nous avons désignées précédemment par le vocable de "condition". Elles s'emploient fréquemment avec if, while, repeat until, ...Elles fournissent comme résultat VRAI ou FAUX.

Elles font usage :

des symboles <, >, <=, >=, =, <>

des connecteurs AND et OR

de la négation NOT

de parenthèses.

Nous n'entrerons pas pour l'instant dans les détails syntaxiques de l'emploi de ces symboles. Contentons nous de dire que, comme l'emploi des parenthèses est parfois OBLIGATOIRE, il est conseillé d'en placer autant qu'il faut pour que les expressions soient claires et non ambiguës.
En tout cas, lorsque plusieurs comparaisons sont connectées par les mots and ou or, chacune d'elles doit figurer entre parenthèses.

Exemples :

```
if (A<>0) and ((Compteur=-1) or (FF <= NombMax)) then
  ...

while (I <= N) and not (I > MM) do
  ...
```

Les divers symboles de comparaisons pourront s'écrire aussi bien entre des informations numériques qu'entre celles de type chaînes de caractères. L'ordre sous-jacent dans ce dernier cas est celui du dictionnaire (ordre lexicographique), les majuscules précédent les minuscules.
Ainsi :

```
'chat' < 'chien'        est vrai
'Chien' < 'chat'        est vrai
```

----- o -----

FORMATION EN EDUCATION N°5.16

RECHERCHE

PUBLICATIONS DU EFIS

Introduction aux bases de données relationnelles et à dBASE III.

1. Introduction aux bases de données

André DELACHARLERIE

DEPARTEMENT
EDUCATION & TECHNOLOGIE

Facultés Universitaires Notre-Dame
de la Paix B-5000 Namur

INTRODUCTION AUX BASES DE DONNEES RELATIONNELLES et à dBASE III

Fascicule 1 : Introduction aux bases de données

A. Delacharlerie

Version provisoire

INTRODUCTION AUX BASES DE DONNEES RELATIONNELLES ET A dBASE III

Fascicule 1 : Introduction aux bases de données

Avant-propos

Un coup d'oeil sur les rayons des librairies spécialisées en informatique permet de constater que les ouvrages traitant des bases de données en général ou bien de dBASE III ne manquent certainement pas; au contraire, ils sont assez nombreux. Alors, pourquoi y ajouter ce fascicule?

La raison en est fort simple. Si l'on y regarde de plus près, on constate que les documents parlant des théories des bases de données sont généralement très techniques ou font appel à des notions avancées de mathématique. A l'autre bout de la chaîne, de nombreux ouvrages, fort intéressants du reste, présentent des logiciels "Systèmes de Gestion de BaseS de Données" (SGBD) et notamment dBASE III en se limitant exclusivement au "mode d'emploi" du logiciel. Il nous a semblé que pour un utilisateur non spécialiste (professeur, économe ou directeur d'école, indépendant ou administrateur d'une petite entreprise), ces deux types d'ouvrages étaient soit trop difficiles, soit trop limités.

Aussi, notre objectif sera, au long de ces quelques pages, de donner, à côté des outils pratiques (par la découverte de dBASE), quelques notions fondamentales sur les bases de données (BD) en général et tout spécialement sur les bases de données relationnelles qui constituent le type de BD le plus moderne et le plus répandu sur micro-ordinateur. Nous tenterons également de donner quelques règles qui nous aideront à concevoir concrètement une (petite) base de données.

Nous l'avons déjà sous-entendu, ces notes ne s'adressent pas aux spécialistes des BD, mais à tout qui désire pouvoir prendre un peu de recul vis-à-vis du logiciel qu'il utilise ou qu'il désire employer par la suite. En conséquence, notre approche sera surtout composée d'exemples et de dessins sans, bien souvent, faire "le tour du problème". Nous désirons surtout que le lecteur découvre le monde des BD et non pas qu'il en subisse la rigueur. Les informaticiens nous excuseront donc pour quelques raccourcis et quelques simplifications dans les théories consacrées.

Néanmoins, nous avons tenu à conserver la terminologie propre aux bases de données de façon à ce que le lecteur ne soit pas désemparé lorsque, à la recherche d'un complément d'information, il consultera l'un ou l'autre des ouvrages présentés dans la bibliographie. Tous ces termes, souvent écrits en caractère gras, sont expliqués soit immédiatement à l'endroit où ils sont introduits, soit un peu plus loin dans un contexte plus favorable à leur bonne compréhension.

Le développement, dans la seconde moitié des années soixante, des supports à accès aléatoire (*) tel que le disque et le tambour magnétiques a rendu possible des organisations de fichiers nouvelles : les données ne doivent plus être stockées séquentiellement (c'est-à-dire l'une à la suite de l'autre), mais peuvent être réparties sur l'entièreté du support. On parle alors de fichiers à accès direct, à accès séquentiel indexé ou à accès calculé Dans ces différents types de fichiers, chaque information est repérée par un numéro que l'on appelle son adresse (comme les différentes maisons d'une rue sont distinguées par leur numéro). L'accès à une information particulière se réalise soit en donnant directement le numéro de localisation (accès direct), soit en retrouvant ce numéro grâce à des tables de correspondance (accès indexé), soit encore en obtenant ce numéro par un calcul mathématique (accès calculé).

La puissance des moyens informatiques allant grandissant, il est tentant de multiplier les programmes d'applications et, partant, de multiplier aussi le nombre de fichiers de données. Chaque service de l'entreprise désire profiter des facilités offertes par l'informatique et fait donc rédiger ses propres programmes et créer ses propres fichiers d'informations.

Cette explosion des fichiers de données amène les informaticiens du début des années septante à prendre conscience d'une série de problèmes. Entre autres, il faudrait, au sein d'un même organisme, limiter, voire supprimer la redondance de certaines données consécutives à la multiplicité des fichiers employés. En effet, si une même information, l'adresse de Monsieur Dupond, par exemple, est stockée dans plusieurs fichiers, utilisés par des services différents, la mise à jour par l'un des services, lors d'un changement d'adresse, ne sera pas nécessairement répercutée dans tous les fichiers où cette adresse se trouve. On dira que le système d'information n'est plus **cohérent** ou qu'il a perdu son **intégrité**. Corrolairement, si l'on limite le nombre de fichiers, il est important de les rendre accessibles par plusieurs applications et même, il devient intéressant de connecter plusieurs fichiers comportant des informations complémentaires.

Les années septante vont donc être le témoin de la naissance des premiers systèmes de gestion de bases de données (SGBD). Il s'agit d'ensembles logiciels capables de prendre en compte la création des structures de données, l'installation des premières informations, puis ensuite de la gestion des données (ajout, suppression, mise à jour, recherche d'informations). Ils se distinguent particulièrement de ce qui

(*) Aléatoire ne signifie bien entendu pas que l'on va chercher les informations au hasard, mais que l'on peut accéder aussi facilement au début, au milieu ou à la fin d'un fichier, ce qui n'était pas vrai avec les bandes magnétiques.

s'était fait jusqu'alors en ce sens qu'ils ne sont pas liés à des applications spécifiques mais qu'ils servent d'intermédiaires entre d'une part les programmes d'application et d'autre part le système d'exploitation (*) et les unités de stockage de l'information (cfr fig. 1). Ces SGBD fonctionnent essentiellement sur de gros ordinateurs et sont à même de répondre, simultanément (ou presque), aux demandes concurrentes de plusieurs utilisateurs. Les données sont organisées suivant des **structures hiérarchiques** (par exemple, système IMS développé par IBM) ou **réseau** (suivant le modèle défini par CODASYL [Conference On Data System Language]). Ces systèmes furent (et sont toujours) largement utilisés. Néanmoins, ils souffrent encore d'au moins deux défauts. Premièrement, la structure de ces bases de données est fort rigide et permet peu d'évolution; ensuite, ces systèmes ne peuvent être utilisés que par l'intermédiaire de programmes d'application. Il est donc impossible à l'utilisateur non spécialiste d'interroger directement la base de données, et moins encore de la créer ou d'en modifier l'organisation.

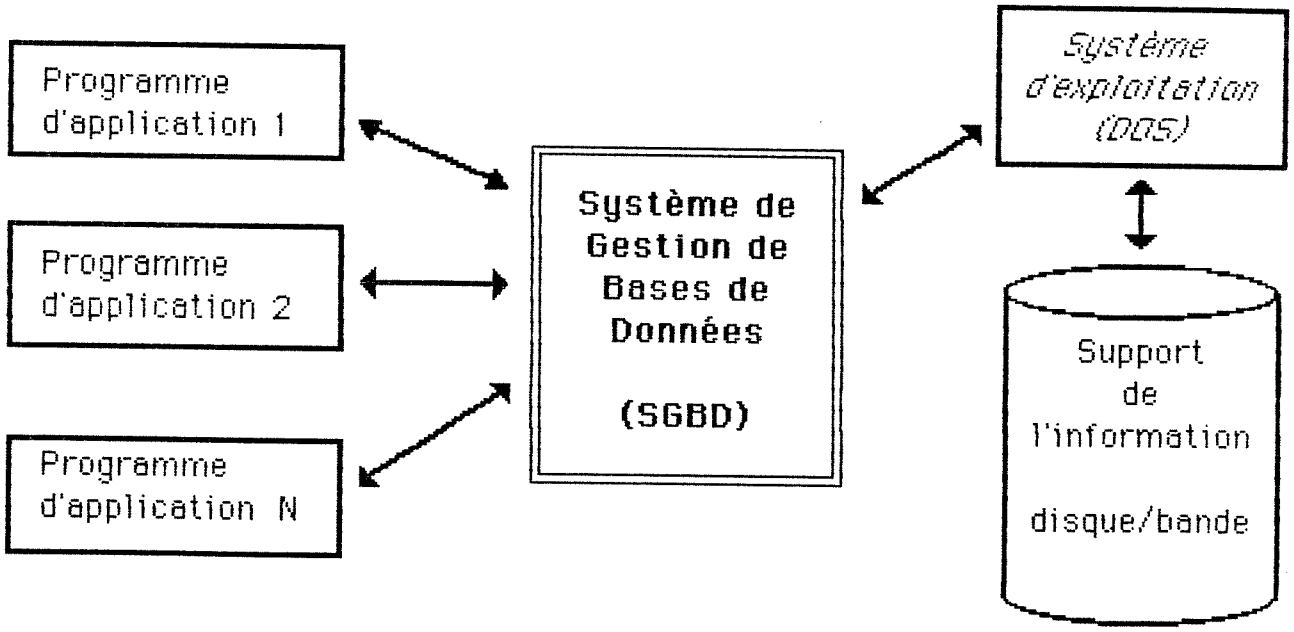


Figure I-1.
Relations du SGBD avec son environnement

Les travaux de E.F. CODD au San Jose Research Laboratory d'IBM vont ainsi donner naissance, vers la fin des années septante, aux **bases de données relationnelles**, qui seront beaucoup plus souples d'emploi et plus accessibles pour l'utilisateur final. De nombreux SGBD seront développés sur ce modèle, citons entre autres SQL/DS et DB2 d'IBM, ORACLE, UNIFY, ...

(*) Le système d'exploitation est le programme qui, entre autres, supervise les échanges entre le coeur du micro-ordinateur (micro-processeur, mémoire centrale) et tous les périphériques (clavier, écran, imprimante et surtout mémoire secondaire sur disque ou bande magnétique).

A cette même époque, l'informatique subit, par ailleurs, une autre révolution de taille : l'arrivée des micro-ordinateurs. Pour ces derniers, on développera tout d'abord des **logiciels de gestion de fichiers** qui ne peuvent, vu l'exiguité de la mémoire centrale ainsi que la faible capacité des disquettes, manipuler qu'un seul fichier à la fois. Le logiciel PFS (rebaptisé Filling Assistant sur les PC compatibles) est très représentatif de cette classe de programmes. Très vite, cependant, certaines sociétés vont chercher à appliquer aux micro-ordinateurs les résultats des recherches sur les bases de données relationnelles : ce sera la naissance par exemple de dBASE II d'Ashton Tate et de KNOWLEDGEMAN de Micro Data Base System. dBASE II doit son succès au fait qu'il offre, sur des micro 8 bits, la possibilité de gérer simultanément deux fichiers de données mais aussi, et peut être surtout, aux deux modes d'utilisation du logiciel. L'utilisateur peut soit donner ses ordres au fur et à mesure par le biais de commandes directes, soit rédiger de véritables petits programmes qui pourront automatiser certaines tâches et rendre l'emploi de la base de données encore plus simple pour l'utilisateur non-spécialiste.

Enfin, l'arrivée des micro-ordinateurs 16 bits à permis d'améliorer considérablement les performances des petites bases de données et a notamment donné naissance à dBASE III que nous étudierons en détail dans les fascicules suivants.

II. **Notion de base de données**

Les expressions fichier, base de données, et banque de données sont aujourd'hui souvent employées. Il semble cependant que l'utilisateur moyen ne sache pas souvent ce qui différencie ces notions. Sans vouloir en donner des définitions magistrales, nous allons essayer de distinguer plus clairement ces différents concepts.

Le mot **fichier** est probablement l'un des plus utilisés par les informaticiens et celui qui recouvre le plus d'"objets" différents. Le point commun de tous ces fichiers, c'est d'être une collection d'informations (au sens le plus large) que l'on peut stocker en utilisant un support magnétique comme une disquette par exemple. Tous, en effet, sont gérés par le système d'exploitation (DOS) qui règle les échanges entre la mémoire centrale du micro-ordinateur et les mémoires auxiliaires constituées le plus souvent, par des disquettes magnétiques ou des disques durs. Hormis cette ressemblance de contenant, les contenus des fichiers manipulés par le micro-ordinateur peuvent être très différents. On y trouve en effet, côte à côte, des **fichiers de données**, comme ceux de dBASE, des **fichiers textes** comprenant des programmmes rédigés dans l'un ou l'autre langage, des **fichiers code** contenant des programmes exécutables en langages machines, et bien d'autres encore. Ces différents types de fichiers ne doivent pas être confondus. A cette fin, le système d'exploitation MS-DOS offre la possibilité de compléter les noms des fichiers par un suffixe en trois lettres. Ces **fichiers de données** en dBASE reçoivent ainsi le suffixe .DBF tandis que les **fichiers de commandes** seront suivis de .PRG; enfin, les **fichiers exécutables** comme le logiciel dBASE lui-même seront affectés du suffixe .EXE ou .COM.

A partir de maintenant, nous nous intéresserons plus spécialement aux fichiers de données. Dans la plupart des cas, ils sont organisés d'une façon qui rappelle assez bien les fichiers manuels que l'on peut réaliser avec des cartons et une boîte. S'il s'agit par exemple d'un fichier d'adresses, toutes les fiches cartonnées sont organisées suivant un même modèle et comprennent une série de rubriques (par exemple, Nom, Prénom, Rue, N° de téléphone, ...) qui peuvent d'ailleurs être préimprimés sur le carton. Chaque fiche contient les informations relatives à un élément de l'ensemble représenté (une personne, membre d'une association, par exemple). Souvent aussi, les fiches ne sont pas rangées au hasard dans la boîte, au contraire, elles sont classées suivant l'ordre croissant (ou décroissant) d'une rubrique (par exemple, l'ordre alphabétique des noms des personnes). Ces fichiers de données manipulés par l'ordinateur ont dans la plupart des cas une structure semblable. Ceux-ci sont en effet souvent composés d'une suite ordonnée d'enregistrements (record) (=fiches) eux-mêmes décomposables en une série de champs (fields)

(=rubriques). Les enregistrements peuvent aussi être rangés suivant un ordre dépendant d'une rubrique mais sont généralement organisés selon d'autres critères (par exemple, l'ordre chronologique de création) pour simplifier et accélérer la gestion du fichier.

Une **base de données**, par contre, peut être définie comme un exemple structuré et cohérent d'informations, accessibles par un ou plusieurs utilisateurs, sur lesquelles on peut développer différentes applications. Cette définition ne fait expressément pas référence aux fichiers car une base de donnée peut, suivant les SGBD, être répartie sur un seul ou plusieur fichiers.

Dans le cadre des micro-ordinateurs, les bases de données que nous développons seront essentiellement mono-utilisateur, bien que, par exemple, dBASE III-plus offre des possibilités d'utilisations concurrentes d'une base de données par plusieurs personnes.

La distinction entre base de données et banque de données est un peu plus floue. On associe le plus généralement le terme de Base de Données à une source centralisée d'informations, interne à une entreprise ou un organisme, alors qu'on appelle **Banque de Données** une source externe accessible par plusieurs entreprises. L'objectif d'une Banque de Données est donc souvent de rassembler de la façon la plus exhaustive possible toutes les informations disponibles sur un domaine de connaissance. De plus, la nature des informations qui y sont gérées est plus diversifiée : cela peut aller de références bibliographiques à des images digitalisées en passant par des textes juridiques. Enfin, la consultation d'une Banque de Données, moins structurée qu'une Base de Données, se réalisera souvent par usage de mots-clés.

III. Objectifs fondamentaux des bases de données

1. Limiter la redondance en centralisant l'information

L'utilisation de fichiers spécifiques à chaque application conduisait à reproduire en plusieurs exemplaires et sur plusieurs supports la même information. Des modes de codage et d'organisation différents rendaient de plus la comparaison ou la connexion de ces fichiers difficile voire impossible.

Le but d'une base de données est au contraire de centraliser l'information et, dans la mesure du possible, de ne stocker qu'une seule fois une même information. On peut ainsi obtenir un gain de place sur les supports et un gain de temps lors de la saisie et de la mise à jour de l'information.

2. Assurer l'intégrité, la cohérence et la fiabilité de l'information

La redondance étant (presque) entièrement éliminée, il devient beaucoup plus simple de vérifier lors de la saisie ou des mises à jour que l'information est "vraisemblable". C'est ce que l'on appelle les **contraintes d'intégrité** de la base de données qui peuvent être de plusieurs types :

- appartenance à une liste de valeurs ou à un intervalle (par exemple, un élève ne peut être inscrit que dans une des classes organisées par l'école);
- répondre à un format particulier (par exemple, n° de téléphone);
- être cohérente avec d'autres informations (par exemple, un élève âgé de 12 ans ne peut se trouver en 4ème année du secondaire).

3. Assurer l'indépendance entre les données et les traitements

La structure d'une base de données doit pouvoir être modifiée sans que les programmes d'application ne doivent être modifiés. Ces changements de structures peuvent être, entre autres, l'ajout d'une rubrique (par exemple, modifier la structure de la base pour qu'elle accepte un nom de jeune fille, alors que cette rubrique n'avait pas été prévue au début) ou la modification de la méthode d'accès à un type d'information visant notamment à accélerer les recherches sur base de tel ou tel critère. Nous verrons que cet objectif est très difficile à atteindre et que les bases de données pour micro-ordinateurs comme dBASE III n'y arrivent qu'imparfaitement.

4.Assurer la sécurité des données

Aucun système informatique n'est à l'abri d'un incident (arrêt sur coupure de l'alimentation électrique, défaillance des supports magnétiques, commande erronnée de l'opérateur, ...). Il est donc important de disposer de moyens pour rétablir la base de données dans un état cohérent après un tel incident. A cet effet, les gros systèmes de gestions de base de données réalisent eux-mêmes des copies de sécurité ("back up") du contenu de la base ainsi qu'un Journal des Transactions (*) reprenant la liste des ajouts et mises à jour effectués depuis la dernière copie de sécurité. Il est ainsi possible de ramener la base de données dans l'état où elle se trouvait juste avant l'incident.

Les petits systèmes, par contre, ne sont en général pas pourvus de tels dispositifs et il appartient donc à l'utilisateur de réaliser lui-même régulièrement des copies de sécurité.

5.Assurer la confidentialité des informations

Cet objectif, comme le précédent, n'est généralement atteint que par les systèmes importants et n'a par ailleurs de réelle utilité que dans le cas de bases de données partagées entre plusieurs utilisateurs. Il s'agit de limiter l'accès à certaines parties des informations suivant les utilisateurs et cela séparément en ce qui concerne la consultation et la mise à jour. Certaines personnes peuvent en effet être autorisées à consulter certaines données sans pour autant avoir le droit de les modifier, de les supprimer ou d'en ajouter de nouvelles.

6.Permettre un partage harmonieux des données

Lors de la manipulation simultanée d'une base de données par plusieurs utilisateurs, il est possible que deux d'entre eux lancent des ordres contradictoires de mise à jour sur une donnée unique, le système doit pouvoir arbitrer ces transactions et verrouiller l'accès à une donnée pendant sa mise à jour. D'autre part, certaines transactions ne peuvent être interrompues sous peine d'introduire des incohérences dans la base. Par exemple, si dans une comptabilité, on effectue un transfert d'un compte vers un autre, il est primordial que les deux opérations de débit et de crédit ne puissent être effectuées l'une sans l'autre, si un incident survient.

(*) On appelle transaction tout échange d'information entre l'utilisateur et la base de données.

IV. Trois modèles de bases de données

Comme nous l'avons vu dans l'introduction, on peut aujourd'hui classer les bases de données actuelles en trois grandes familles : les bases de données - hiérarchiques;
- en réseau;
- relationnelles.

Plutôt que de décrire techniquement chacune de ces organisations nous allons les présenter en partant d'un exemple simple pour lequel nous proposerons une implémentation dans chaque organisation.

1. Les bases de données hiérarchiques

Ce modèle, le plus ancien, postule une organisation arborescente de l'information. Dans l'exemple de la figure IV-1, on peut voir que chacune des classes d'une école est reliée à un certain nombre de professeurs, eux-mêmes reliés à une série de cours. Par ailleurs, la classe est également reliée à une série d'élèves.

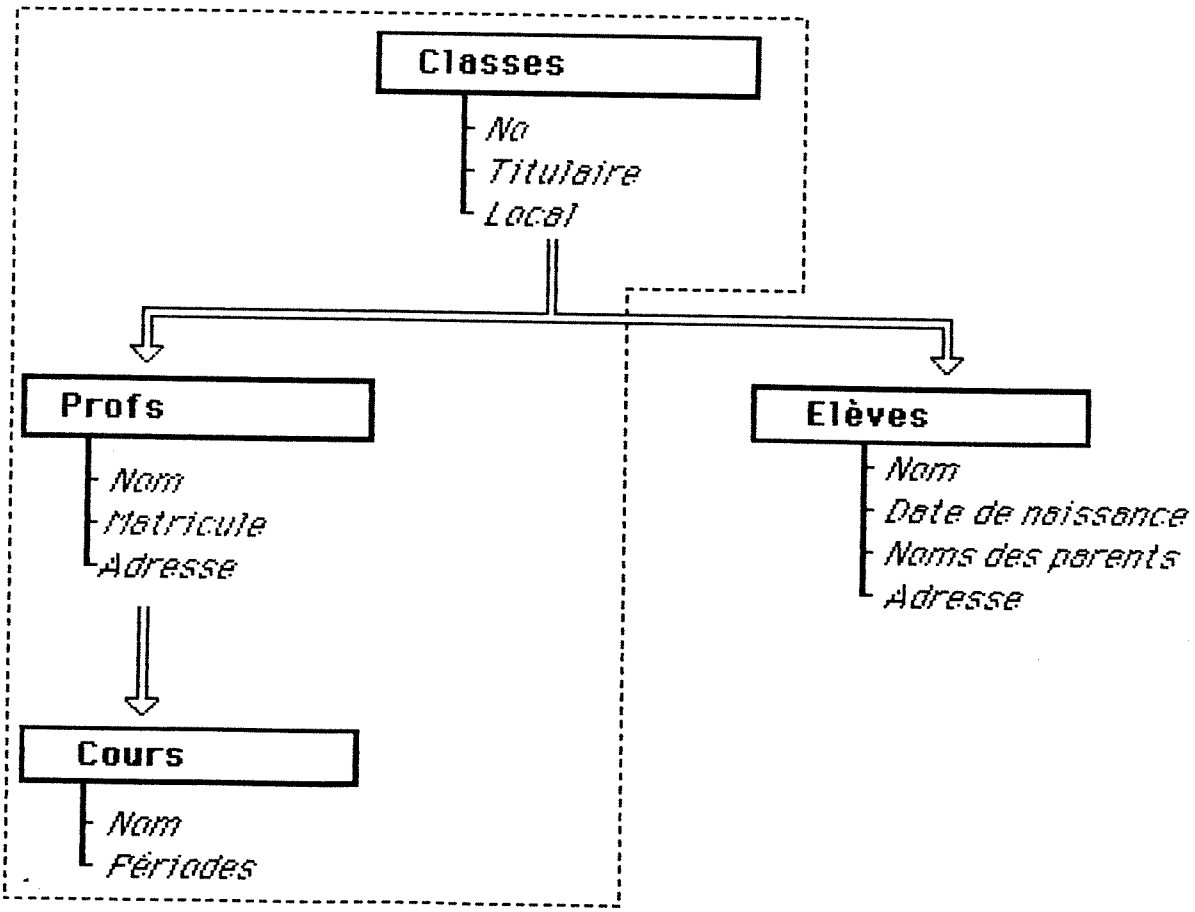


Figure IV-1.
Schéma hiérarchique

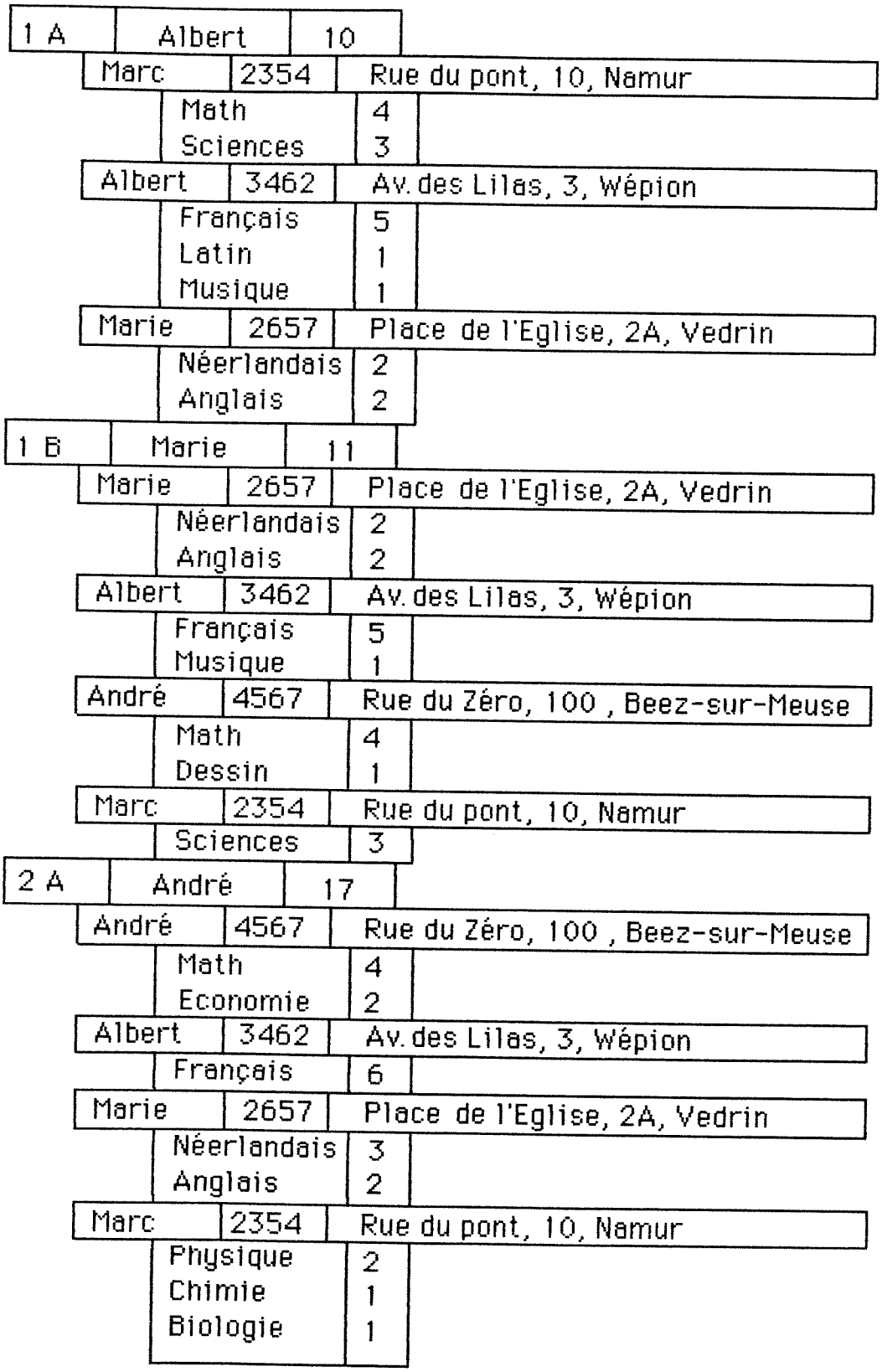


Figure IV-2.
Occurences hiérarchiques

Pour fixer les idées, nous donnons à la figure suivante trois **occurences hiérarchiques** de la partie encadrée par une ligne tiretée du **schéma** ci-dessus. Ce schéma fait clairement apparaître quatre **types d'entités** : Classes, Profs, Cours, Elèves, ainsi que trois **liaisons** ou **types d'associations**, illustrées par les flèches, et qui représentent la dépendance qui existe entre, par exemple, une classe et les professeurs qui enseignent dans cette classe. Ce genre de liaison est dit de type **1-N** ou **un à plusieurs** car, sauf exceptions, chaque association lie une entité "père" à plusieurs entités "fils". Une classe est ainsi associée à plusieurs professeurs et un professeur est normalement lié à plusieurs cours.

On voit sur la figure IV-2 que ces associations sont représentées implicitement par la juxtaposition des **enregistrements** "père" et "fils" dans un même **segment** (*) de données. Cette représentation a évidemment l'avantage de la simplicité mais est par contre d'une grande rigidité. Un des gros défauts de ce type de base de données provient de la nécessité de dupliquer certaines informations. On voit en effet que l'adresse d'un professeur enseignant dans les trois classes est recopiée trois fois dans la base de données.

Notons enfin que l'interrogation d'une telle base nécessite la rédaction d'une procédure indiquant au SGBD, la marche à suivre pour obtenir l'information. Prenons deux exemples :

a) Qui enseigne en 1B ?

La marche à suivre peut être la suivante (**) :

Chercher le segment concernant la 1B
Pour chaque occurence de professeur
Afficher le nom du professeur

(*) Segment désigne dans la terminologie proposée par IBM pour son SGBD IMS, le "paquet d'information" formé par chaque occurence hiérarchique, c'est-à-dire l'ensemble des informations liées à une même "racine d'arbre".

(**) Nous avons volontairement simplifié la description de ces algorithmes (ou marches à suivre) pour les rendre accessibles aux lecteurs non programmeurs.

b) Quels cours sont donnés par André ?

La marche à suivre est ici plus complexe, car il nous faut balayer tous les segments classe pour être sûr de n'oublier aucun cours :

Pour chaque classe
Chercher le professeur dont le nom est "André"
Si "André" est présent, alors
Pour chaque cours
Afficher le nom du cours

On notera que ces algorithmes sont très dépendants de la structure de la base de données : si la structure change, les algorithmes devront eux-mêmes être réécrits.

2. Les bases de données en réseau

Nous venons de voir que le modèle hiérarchique souffrait de deux défauts importants : la rigidité des liaisons entre les entités et la nécessité de dupliquer certains informations. Ainsi, dans le but de remédier à ces problèmes, le comité CODASYL (Conference On Data System Language) a défini en 1971 un nouveau modèle de base de données : le modèle réseau. Comme on peut le voir sur le schéma de la figure IV-3, il est à présent possible de créer un réseau de liaisons entre les différentes entités de la base. De plus, chaque entité ne sera plus présente qu'une seule fois dans la base comme le montre la figure IV-4. L'adresse du professeur "Marie" n'est ainsi plus donnée qu'une seule fois. Cette dernière figure illustre clairement le fait qu'une base de donnée en réseau est constituée de deux sortes d'objets différents : d'une part, des **RECORD** reprenant la description des **entités** de la base, et d'autre part, des **SET** qui décrivent quant à eux diverses **liaisons** pouvant exister entre les entités. Chaque SET est composée d'un **propriétaire (OWNER)**, encadré dans notre figure, et d'un certain nombre (0, 1 ou plusieurs) de **membres (MEMBERS)** qui sont ainsi associés au propriétaire du SET. Il est à noter que, au niveau de l'implémentation physique, les SET ne reprennent pas, comme sur notre figure, des copies des informations des entités, mais sont constitués de **pointeurs** (*) vers les enregistrements concernés.

(*) Un pointeur (vers une information) est un nombre qui donne la localisation (adresse) de l'information pointée. Grâce à cette adresse, on peut retrouver extrêmement rapidement cette information.

On comprend aisément qu'il devient ainsi possible de créer autant d'associations entre les entités qu'on le désire. De plus, ces liaisons ne sont plus obligatoirement du type 1-N, mais peuvent aussi être de type **N-1 (plusieurs à un)** ou **N-M (plusieurs à plusieurs)**.

Dans le schéma présenté en figure IV-3, on trouve ainsi trois liaisons de types N-M : Classes-Profs, Profs-Cours et Classes-Cours. En effet, pour prendre un exemple, dans la liaison Classes-Profs, chaque classe compte plusieurs professeurs, mais chaque professeurs peut aussi enseigner dans plusieurs classes. La liaison Classes-Elèves reste par contre du type 1-N, un élève ne pouvant être inscrit , en principe, dans plusieurs classes. Enfin, on peut voir que le type d'entité élève a éclaté en deux types d'entités : Elèves et Parents, reliées par une liaison de type N-1 : plusieurs élèves de l'école pouvant avoir les mêmes parents mais pas inversément; un élève ne peut avoir plusieurs (couples de) parents.

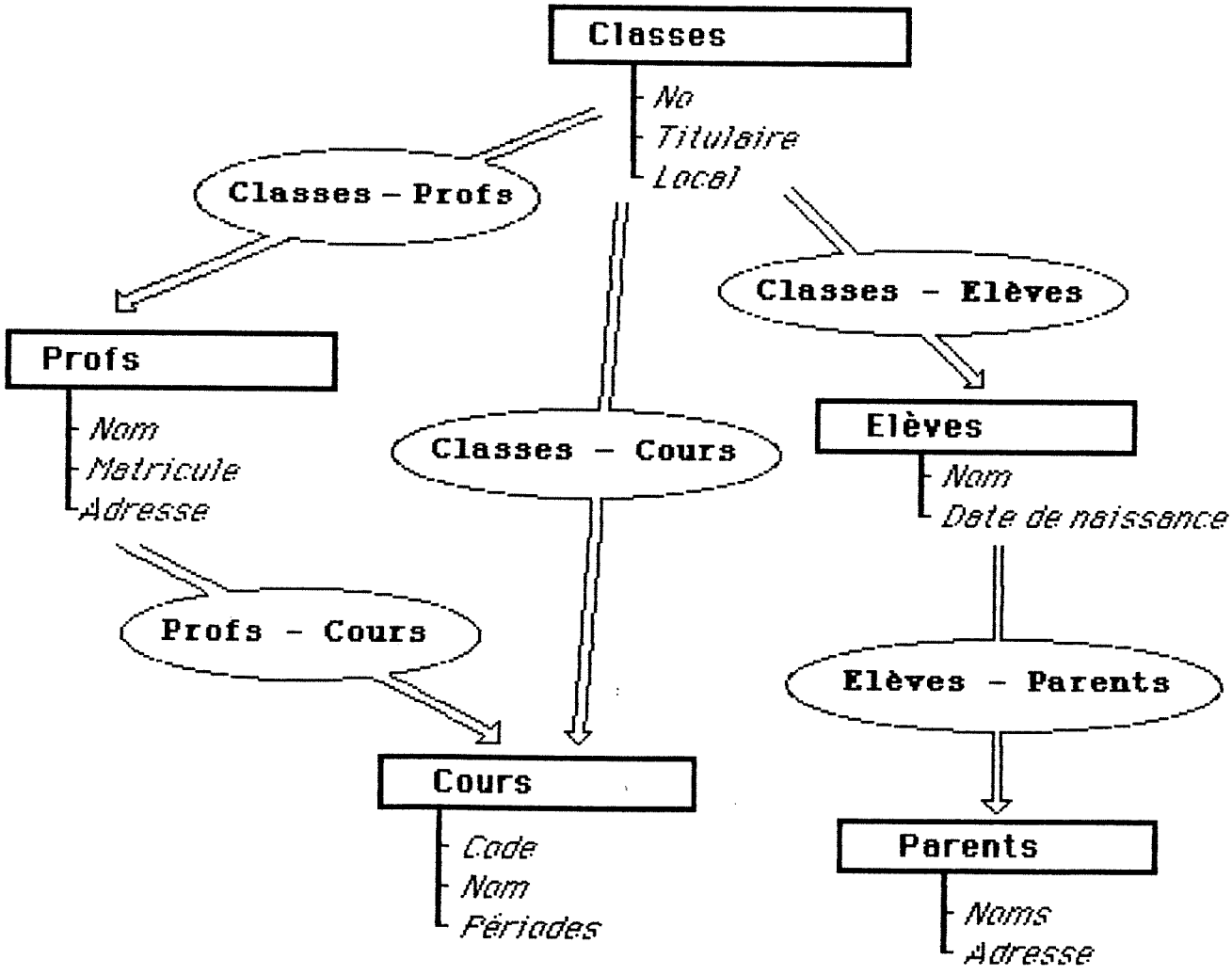


Figure IV-3.
Schéma réseau

1 A	Albert	10
1 B	Marie	11
2 A	André	17

Albert	3462	Av. des Lilas, 3, Wépion
André	4567	Rue du Zéro, 100 , Beez-sur-Meuse
Marie	2657	Place de l'Eglise, 2A, Vedrin
Marc	2354	Rue du pont, 10, Namur

C1	Français	5
C2	Français	6
C3	Math	4
C4	Sciences	3
C5	Physique	2
C6	Chimie	1
C7	Biologie	1
C8	Latin	1
C9	Néerlandais	2
C10	Néerlandais	3
C11	Anglais	2
C12	Musique	1
C13	Dessin	1
C14	Economie	2

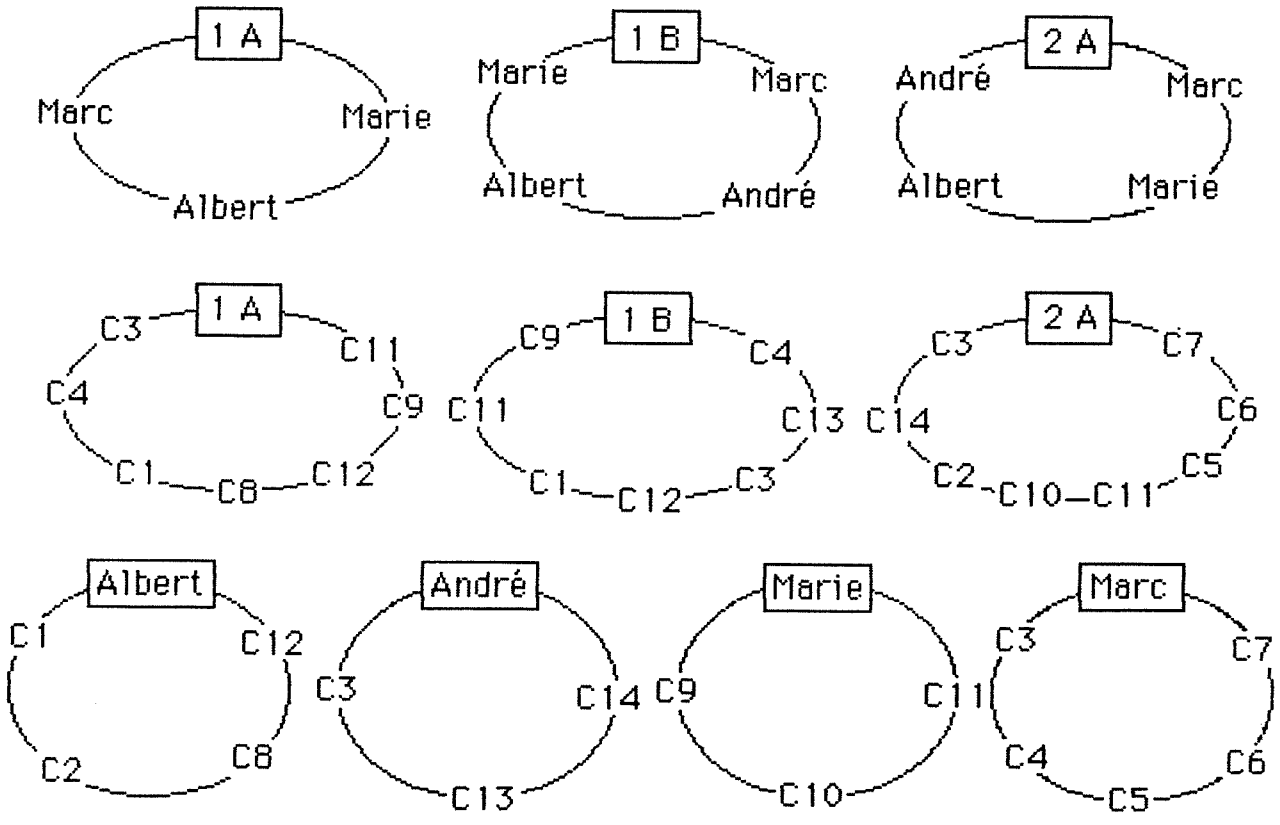


Figure IV-4.
Occurences réseau

Au niveau de l'interrogation d'une telle base de donnée, on notera que celle-ci est en général simplifiée par rapport au modèle hiérarchique. Elle nécessite cependant toujours de connaître la structure de la base. La réponse aux questions posées précédemment nécessite aussi la rédaction de procédures :

a) Qui enseigne en 1B ?

Chercher la classe "1B"
Répète
Chercher le professeur (suivant) associé à cette classe par le SET Classes-Profs
Afficher le nom du professeur
Jusqu'à ce qu'il n'y ait plus de professeur dans ce SET

b) Quels cours sont donnés par André ?

La procédure est semblable à celle ci-dessus, il suffit ici de chercher le RECORD "André", puis d'accéder aux différents cours via le SET Profs-Cours.

On notera qu'il est également possible, connaissant un membre d'un SET, d'obtenir le propriétaire du SET. Cela serait nécessaire, par exemple, si l'on cherchait à obtenir les classes où un cours de Dessin est donné.

Remarque : Le lecteur attentif notera que les schémas IV-1 et IV-3 ne sont pas **équivalents**. En effet, la question "Qui enseigne le cours de math en 1B ?" ne peut plus trouver de réponse précise dans l'organisation réseau proposée. L'équivalence des schémas peut, bien entendu, être rétablie mais cela nous aurait obligé à compliquer la structure sans apporter plus de clarté à notre propos.

3. Les bases de données relationnelles

Le modèle relationnel a été élaboré, sur base des théories mathématiques de l'algèbre relationnelle, dans le but d'arriver à une meilleure indépendance entre la structure de la base de données et les procédures d'accès aux informations. Ce modèle devait, de plus, être suffisamment simple pour permettre à un utilisateur non programmeur d'interroger et de modifier la base de données.

L'élément essentiel de ce modèle est la **table**. Chacun des types d'entités que nous avons vue peut en effet être représentée, comme le montrent les figures IV-5 et IV-6, par une table où chaque champ correspond à une **colonne** et chaque entité occupe une **ligne**. Les associations sont elles aussi représentées dans ce modèle en reprenant certains champs dans plusieurs tables avec cependant l'inconvénient d'introduire une certaine redondance. Heureusement cette dernière peut souvent être gérée assez facilement.

La représentation tabulaire est manifestement simple et peut être utilisée facilement par tous. Elle repose néanmoins sur des bases mathématiques solides et comporte bien entendu un "jargon" spécifique dont voici quelques termes. Chaque table correspond ainsi à une **relation** entre plusieurs ensembles de valeurs (appelés **domaines**). La relation est un ensemble de **n-uples** ou **n-uplets** (couples, triplets, quadruplets ...) matérialisés chacun par une ligne de la table. Les colonnes des tables sont appelées les **attributs** des relations.

Nous noterons aussi que chaque table possède les propriétés suivantes :

- Il n'y a pas deux lignes identiques;
- L'ordre des lignes n'est pas significatif;
- L'ordre des colonnes n'est pas significatif.

Il faut donc se garder de confondre la notion de table avec celle de tableau telle qu'elle est définie dans la plupart des langages de programmation; là en effet, l'ordre des lignes et des colonnes est important tandis qu'il est possible d'avoir plusieurs lignes aux contenus identiques.

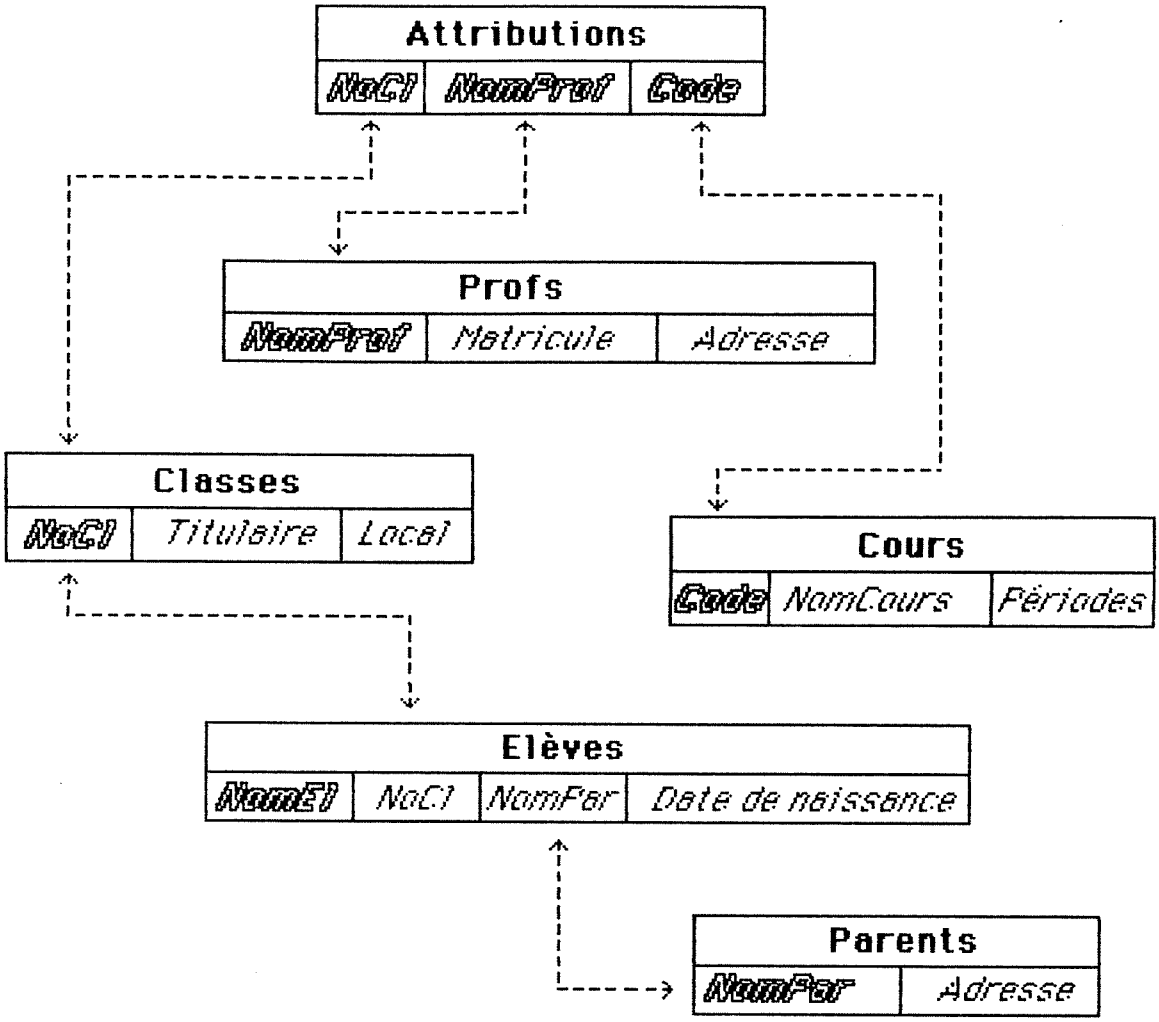


Figure IV-5.
Structures relationnelles

Une autre caractéristique importante des tables employées dans le modèle relationnel, c'est de toujours comporter ce que l'on appelle une **clé primaire**. Cette clé est composée d'une ou plusieurs colonnes dont les valeurs permettent d'identifier chaque ligne. Dans nos schémas, les noms des colonnes formant les clés sont écrits en double trait. NoCl est ainsi la **clé simple** de la table classe. Attributions, par contre, possède une **clé composée** formée par les attributs NoCl, NomProf et Code. Nous verrons plus loin que la présence et le choix de la clé primaire sont très important pour éviter d'introduire des redondances superflues dans la base de données.

L'interrogation d'une base de données relationnelle peut être réalisée soit par l'utilisation de l'**algèbre relationnelle**, soit du **calcul relationnel**, soit encore par la rédaction de **procédures**. Nous développerons les principales opérations de l'algèbre relationnelle dans le chapitre suivant, et nous montrerons comment elle peut être mise en oeuvre dans un langage relationnel tel que SQL ainsi que dans le langage de dBASE III. Le calcul relationnel, aussi baptisé calcul des prédicats,

Profs		
NomProf	Matricule	Adresse
Albert	3462	Av. des Lilas, 3, Wépion
André	4567	Rue du Zéro, 100 , Beez-sur-Meuse
Marie	2657	Place de l'Eglise, 2A, Vedrin
Marc	2354	Rue du pont, 10, Namur

Classes		
NoCl	Titulaire	Local
1 A	Albert	10
1 B	Marie	11
2 A	André	17

Cours		
Code	Nom	Périodes
C1	Français	5
C2	Français	6
C3	Math	4
C4	Sciences	3
C5	Physique	2
C6	Chimie	1
C7	Biologie	1
C8	Latin	1
C9	Néerlandais	2
C10	Néerlandais	3
C11	Anglais	2
C12	Musique	1
C13	Dessin	1
C14	Economie	2

Attributions		
NoCl	NomProf	Code
1A	Marc	C3
1A	Marc	C4
1A	Albert	C1
1A	Albert	C8
1A	Albert	C12
1A	Marie	C9
1A	Marie	C11
1B	Marie	C9
1B	Marie	C11
1B	Albert	C1
1B	Albert	C12
1B	André	C3
1B	André	C13
1B	Marc	C4
2A	André	C3
2A	André	C14
2A	Albert	C2
2A	Marie	C10
2A	Marie	C11
2A	Marc	C5
2A	Marc	C6
2A	Marc	C7

Figure IV-6.
Représentation tabulaire des relations

est par contre plus difficile à aborder, car il s'appuie plus largement sur une approche mathématique. Nous ne le présenterons donc pas dans ces notes, d'autant plus qu'il est moins souvent disponible sur les SGBD classiques. L'interrogation par le biais de procédures est moins typique des BD relationnelles mais permet dans certains cas d'accélérer considérablement la réponse à des requêtes spécifiques. Nous verrons de nombreux exemples de procédures dans le fascicule 3 consacré à la programmation en dBASE.

V. Opérations sur les bases de données relationnelles

L'utilisation d'une base de données, que son modèle soit hiérarchique, réseau ou relationnel, peut être scindée en deux phases : tout d'abord, la description de la structure de la base et ensuite la manipulation et la mise à jour des données. Dans les systèmes hiérarchiques et réseaux, ces deux phases sont très clairement séparées et font généralement l'objet de deux langages distincts, respectivement appelés Langages de Définition de Données (LDD) et Langages de Manipulation de Données (LMD). Dans les SGBD relationnels, par contre, ces deux langages ne forment qu'un et donnent ainsi une bien plus grande souplesse d'utilisation. Il est en effet possible, à tout moment, de créer ou de supprimer une table ainsi que d'ajouter ou enlever un attribut à une relation. Néanmoins, ces modifications de structures ne doivent pas être faites à la lègère car elles peuvent entraîner des pertes de données ou la rupture de l'intégrité de la base. Nous verrons dans le chapitre consacré à la conception de la base de données que le choix de la structure n'est pas chose aisée et doit, si l'on veut éviter des problèmes d'utilisation, répondre à une série de règles.

1. Définition de la base de données

Voyons à présent comment se réalise la définition de la BD dans un SGBD de type relationnel. Bien que cette série de fascicules s'intéresse plus spécifiquement à dBASE III, nous donnerons aussi quelques exemples de commandes rédigées dans le langage SQL (Structured Query Language) associé au SGBD ORACLE dont une version est disponible sous MS-DOS. Nous pensons en effet qu'il est intéressant que le lecteur se familiarise dès à présent avec SQL, car d'une part, ce langage a été retenu par l'institut américain des normes ANSI comme le langage standard pour les SGBD relationnels et d'autre part, on sait déjà, au moment de la rédaction de ces notes, que la prochaine version du logiciel vedette d'Ashton-Tate, probablement nommée dBASE IV, sera capable de traiter des requêtes formulées en SQL.

En ce qui concerne dBASE III, nous ne donnerons ici que des exemples d'instructions valides, sans entrer dans les détails de syntaxe; pour une description technique des commandes, on se reportera au fascicule 2, consacré à l'utilisation interactive du logiciel.

Créer une BD relationnelle, c'est essentiellement créer des tables en en décrivant les attributs (colonnes). Les tables peuvent être créées dans n'importe quel ordre. La base de données étant l'ensemble des tables accessibles à un instant déterminé.

Créons par exemple, la table Classes que nous avons illustrée à la figure IV-6.
En SQL, on donne la commande :

```
create table CLASSES    ( NOCL char (2),
                          TITULAIRE char (15),
                          LOCAL number (3,0) )
```

La table créée portera donc le nom CLASSES et comportera trois attributs. NOCL, qui pourra contenir 2 caractères, TITULAIRE, qui pourra en contenir 15, et LOCAL qui sera formaté pour recevoir des nombres de 3 chiffres avec zéro décimale.
En dBASE, la commande de création d'une table (c'est-à-dire aussi d'un fichier) est interactive : on frappe d'abord :

```
create CLASSES
```

puis on décrit les différentes colonnes (appelées champs par dBASE) en répondant aux questions qui apparaissent à l'écran. Le résultat que l'on obtient avec l'exemple ci-dessus est le suivant :

	Nom champ	Type	Dim	Dec
1	NOCL	Caractère	2	
2	TITULAIRE	Caractère	15	
3	LOCAL	Numérique	3	0

Tant en SQL qu'en dBASE, cette création produit une table dont la structure est connue du système mais qui est encore totalement vide.
Le remplissage de la table peut se réaliser de plusieurs façons. En SQL, par exemple, on pourra dire :

```
insert into CLASSES values ('1A', 'Albert', 10)
```

En dBASE, on utilisera souvent la commande interactive :

```
append
```

qui nous permettra d'introduire facilement les données dans chacun des champs.

Un autre aspect de la création de la base de données, c'est la déclaration des index. Sans entrer dans le détail, considérons qu'un index est un "outil" intimement lié à une colonne de la table et qui permet d'accélérer considérablement les recherches où cette colonne intervient. Bien

souvent, on déclarera un index pour la clé de la table mais on peut aussi en créer pour d'autres colonnes. La déclaration d'un index, baptisé NOCLASSE, associé à la clé NOCL de la table CLASSE s'obtiendra en SQL par :

create index NOCLASSE on CLASSES (NOCL asc)

et en dBASE, si le fichier CLASSES est déjà ouvert, par :

select CLASSES
index on NOCL to NOCLASSE

2. Interrogation de la base de données

La forme la plus naturelle et la plus classique d'interrogation d'une BD relationnelle consiste utiliser les opérations de l'algèbre relationnelle développée par E.F. CODD. Il en existe trois principales : la projection, la sélection et la jointure, auxquelles on peut ajouter les opérations classiques sur les ensembles, à savoir la réunion, l'intersection et la différence, mais qui sont beaucoup moins employées.

Il s'agit bien d'opérations (ou de fonctions) sur les tables; autrement dit, comme une opération sur des nombres (l'addition par exemple) produit un résultat qui est un nombre aussi, les opérations que nous allons décrire donnent toujours une nouvelle table pour résultat. Néanmoins, cette nouvelle table ne sera, bien souvent, pas stockée dans la base de données car, puisqu'elle peut être obtenue à partir des tables de base, ses informations sont donc automatiquement redondantes.

D'une façon générale, ces opérations se traduisent par des variantes d'une mêmes commande qui sera **select** en SQL et **list**, **display** ou **copy** en dBASE (ces trois commandes ne se différencient guère que par la façon dont la table résultat est rendue à l'utilisateur).

a) La projection

Un premier type de requête que l'on peut adresser à une BD peut s'exprimer par "Donnez la liste de tous les ... de ...".
Exemples :
- "Donnez la liste de tous les noms et adresses de profs";
- "Donnez la liste de tous les titulaires de classes";
- "Donnez la liste de tous les intitulés de cours".

La réponse à ces requêtes s'obtient en ne retenant que certaines des données de la table concernée. On dit alors qu'on effectue une

projection de table, c'est-à-dire qu'on crée une nouvelle table où l'on ne retient qu'une partie des colonnes mais où toutes les lignes sont conservées. Par exemple, si l'on projette la relation PROFS de la figure IV-6 sur ses attributs NOMPROF et ADRESSE, pour répondre à la première des requêtes ci-dessus, la commande de projection peut s'écrire en SQL :

```
select NOMPROF, ADRESSE
    from PROFS
```

et en dBASE :

```
select PROFS                    (*)
list all fields NOMPROF, ADRESSE
```

NomProf	Adresse
Albert	Av. des Lilas, 3, Wépion
André	Rue du Zéro, 100 , Beez-sur-Meuse
Marie	PLace de l'Eglise, 2A, Vedrin
Marc	Rue du pont, 10, Namur

Figure V-1.
Projection de la table PROFS

b) La sélection

Un second type de requête à une BD s'exprime par une phrase du genre : "Quels sont les ... qui ..."...
Exemples :
- "Quels sont les profs qui enseignent en 1B ?";
- "Quels sont les cours qui ne comportent qu'une période ?";
- "Quelles sont les classes qui ont "Albert" pour titulaire ?.

Pour extraire les réponses à ces requêtes, il suffit d'effectuer une sélection dans la table concernée. La sélection consiste à ne retenir d'une table qu'une partie des lignes; celles qui satisfont à une condition que l'on pose. Par exemple, sélectionnons dans la table COURS tous les cours qui ne comportent qu'une période par semaine.

(*) La commande select en dBASE permet de désigner la table sur laquelle on va travailler, elle peut être omise si la bonne table est déjà active (par exemple, parce qu'on l'a déjà sélectionnée pour une commande antérieure).

On obtient la table suivante :

Code	*Nom*	*Périodes*
C6	Chimie	1
C7	Biologie	1
C8	Latin	1
C12	Musique	1
C13	Dessin	1

Figure V-2.
Sélection dans la table COURS

La commande qui permet cette sélection peut être énoncée en SQL :

```
select *
    from COURS
    where PERIODES = 1
```

Le signe * signifie "toutes les colonnes" mais peut être remplacé par l'énoncé des noms de colonnes.

En dBASE, la même sélection s'obtient par :

```
list all for PERIODES = 1
```

c) *La jointure*

Dans certains cas, une requête peut nécessiter l'examen de plusieurs tables simultanément (et non d'une seule comme ci-dessus).
Exemples :
- "Donnez la liste des cours qui sont donnés par Marc";
- "Quel est le matricule du titulaire de 2A ?".

Pour répondre à ce genre de questions, il faut d'abord "assembler" deux (voire même trois, quatre, ...) tables, puis ensuite effectuer une sélection ou une projection. C'est l'opération de jointure qui permet cet assemblage.

Réaliser une jointure (appelée aussi composition) de deux relations, c'est les rassembler en une seule table et ce en fonction d'une colonne commune. Composons ainsi les tables ATTRIBUTIONS et COURS pour obtenir une liste d'attributions plus claire. La colonne commune est donc CODE.

On obtient la table :

NoCl	NomProf	Code	Nom	Périodes
1A	Marc	C3	Math	4
1A	Marc	C4	Sciences	3
1A	Albert	C1	Français	5
1A	Albert	C8	Latin	1
1A	Albert	C12	Musique	1
1A	Marie	C9	Néerlandais	2
1A	Marie	C11	Anglais	2
1B	Marie	C9	Néerlandais	2
1B	Marie	C11	Anglais	2
1B	Albert	C1	Français	5
1B	Albert	C12	Musique	1
1B	André	C3	Math	4
1B	André	C13	Dessin	1
1B	Marc	C4	Sciences	3
2A	André	C3	Math	4
2A	André	C14	Economie	2
2A	Albert	C2	Français	6
2A	Marie	C10	Néerlandais	3
2A	Marie	C11	Anglais	2
2A	Marc	C5	Physique	2
2A	Marc	C6	Chimie	1
2A	Marc	C7	Biologie	1

Figure V-3.
Jointure des tables ATTRIBUTIONS et COURS

Cette table comporte une série de redondances car des lignes de la table COURS ont dû être reprises plusieurs fois.

L'ordre SQL qui permet d'obtenir cette table est :

```
select *
    from ATTRIBUTIONS, COURS
    where ATTRIBUTIONS.CODE = COURS.CODE
```

Avec dBASE, la même opération est un peu moins simple car il faut d'abord établir un lien (malencontreusement baptisé "relation" par Ashton-Tate) entre les deux tables à assembler.

La séquence de commandes est ainsi :

```
select ATTRIBUTIONS
set relation to CODE into COURS
list fields NOCL, NOMPROF, CODE,
            COURS -> NOM, COURS -> PERIODES
```

d) Combinaisons de ces opérations

Bien souvent, les requêtes que nous voudrons formuler nécessiteront la combinaison d'une sélection, d'une projection et parfois d'une jointure. Par exemple, pour répondre aux questions que nous nous étions posées dans le chapitre IV, on aurait :

*) Donnez la liste des noms des profs qui enseignent en 1B ?
La commande à donner peut être énoncée en français :

Sélectionner les noms de professeurs (projection)
des lignes de la table ATTRIBUTIONS
pour lesquelles la classe est '1B' (sélection)

Ce qui donne en SQL :

```
select NOMPROF
    from ATTRIBUTIONS
    where NOCL = '1B'
```

ou en dBASE :

```
select ATTRIBUTIONS
list fields NOMPROF for NOCL = '1B'
```

et l'on obtient la table :

NomProf
Marie
Marie
Marie
Albert
Albert
André
André
Marc

Figure V-4.

On constate que celle-ci reprend plusieurs fois certains noms. SQL nous offre un moyen simple d'éliminer ces doublons en ajoutant le mot DISTINCT dans la requête :

```
select distinct NOMPROF
    from ATTRIBUTIONS
    where NOCL = '1B'
```

On obtient ainsi la table :

NomProf
Marie
Albert
André
Marc

Figure V-5.

Le logiciel dBASE n'offre pas cette facilité. Nous verrons dans le fascicule 2 comment simuler cette possibilité grâce à un index et à l'option SET UNIQUE.

*) Quels cours sont donnés par André ?
Ici, nous allons employer à la fois la table ATTRIBUTIONS (pour obtenir les codes des cours) et la table COURS (pour avoir l'intitulé complet). Notre requête peut donc s'énoncer :

Afficher
~~Sélectionner~~ les noms et les périodes (projection)
des lignes des tables COURS et ATTRIBUTIONS
pour lesquelles le code est identique (jointure)
et le nom du professeur est André (sélection)

soit en SQL :

```
select NOM, PERIODES
    from COURS, ATTRIBUTIONS
    where COURS.CODE = ATTRIBUTIONS.CODE
      and NOMPROF = 'André'
```

et en dBASE (on suppose que les fichiers COURS et attributions sont ouverts) :

```
select ATTRIBUTIONS
set relation to CODE into COURS
list fields COURS -> NOM, COURS -> PERIODES
    for NOMPROFS = 'André'
```

La table obtenue est la suivante :

Nom	*Périodes*
Math	4
Dessin	1
Math	4
Economie	2

Figure V-6.

Ces quelques exemples montrent bien que l'interrogation d'une base de données relationnelles n'est pas bien compliquée et ne nécessite en tous cas aucune connaissance en programmation.

On aura remarqué à l'occasion de cette petite comparaison entre SQL(*) et dBASE que le logiciel d'Ashton-Tate est largement empreint par le modèle relationnel mais (et nous découvrirons bien d'autres limitations) ne peut pas vraiment être qualifié de Système de Gestion de Bases de Données Relationnelles. Que le lecteur se rassure cependant, les fascicules 2 et 3 lui montreront que dBASE III est déjà un outil merveilleux et qu'il est à même d'apporter des solutions à de très nombreux problèmes de gestion de données.

(*) Nous n'avons montré ici que quelques facettes de ce langage, mais il en compte bien d'autres qui permettront notamment de gérer la confidentialité, la sécurité et le partage des données.

VI. **Normalisation des relations**

La manipulation des données sous la forme de table est, on l'a montré dans les chapitres précédents, la méthode la plus simple et la plus abordable pour l'utilisateur non-informaticien.

Néanmoins, on va voir que la répartition des données entre les différentes tables doit respecter un certain nombre de règles faute de quoi la mise à jour des tables risque d'introduire rapidement des incohérences et, partant, la base de données ne sera plus fiable, donc pratiquement inutilisable. Les règles dont on vient de parler consistent à décomposer les tables par étapes successives (appelées **formes normales**)jusqu'à ce qu'elles soient les plus simples possible, en évitant au maximum les redondances tout en gardant à la base de données la même "signification globale".

La table que nous avons vue à la figure V-3 par exemple, ne convient pas comme table de base car elle n'est pas normalisée. On avait vu que certaines informations y étaient redondantes. Ainsi, on y "apprend" 3 fois que le code C11 fait référence au cours d'Anglais 5 heures. Cette situation n'est pas favorable à une bonne gestion car une modification concernant ce cours doit être répercutée 3 fois. De même, au niveau des suppressions, l'élimination d'une attribution (parce que le cours n'est momentanément plus organisé) a pour conséquence d'éliminer aussi les informations sur ce cours puisque toute la ligne est enlevée. Ces inconvénients sont supprimés si l'on éclate cette table en deux (pour retrouver les relations ATTRIBUTIONS et COURS). On notera cependant que la seconde formule (en deux tables) ne comprend pas plus ni moins d'informations que la première formule ne comportant qu'une seule table. Il en découle que l'essentiel dans une base de données n'est pas seulement d'avoir toutes les informations que l'on désire, mais aussi que ces informations soient convenablement organisées.

1. Clés primaires

Avant de nous attaquer à la normalisation proprement dite des tables, il convient de revenir sur la notion de **clé primaire**. La clé primaire d'une table est formée par un (**clé simple**) ou plusieurs (**clé composée**) attribut(s) de la relation et est telle que ses valeurs permettent d'identifier de manière unique chaque ligne de la table.

Dans la relation ELEVE que voici, NOELEV est la clé primaire simple.

Elève					
NoElev	Nom	D. Naiss.	Adresse	Téléph.	Classe

Figure VI-1.
Clé primaire simple

tandis que dans la relation PROGRAMME, il faut au moins deux attributs (ANNEE et OPTION) pour identifier sans erreur possible le programme d'une classe.

Programme								
Année	Option	Math	Fr	Néerl	An	Bio		Gym

Figure VI-2.
Clé primaire composée

On notera que certaines relations nous offrent plusieurs **clés possibles**. Par exemple, dans la relation ELEVE ci-dessus, l'attribut NOM (éventuellement composé avec l'attribut DATE NAISS) était aussi une clé possible (encore appelée clé candidate). Lorsqu'une relation possède ainsi plusieurs clés possibles, on choisit toujours la plus simple (la moins longue en nombre de caractères) car le fonctionnement du SGBD sera d'autant plus rapide que la clé primaire est courte.

Parfois, si la clé primaire d'une relation est trop longue ou trop complexe, on créera une clé primaire artficielle plus simple : cela explique la prolifération des numéros de codes, de compte, de matricule, ...

La première propriété des valeurs de la clé primaire est évidemment d'être toutes différentes les unes des autres. Il en découle quelques règles qui devraient en principe être toujours respectées :

- la clé doit être définie (c'est-à-dire doit avoir une valeur) pour toutes les lignes de la relation;
- la clé ne peut subir de mise à jour.

2. Règles de normalisation

a) Première forme normale

La première règle de normalisation a pour but d'obtenir des tables rectangulaires telles que celles que nous avons manipulées au chapitre V. La première forme normale (First Normal Form ou 1NF) peut être définie comme suit :

Une relation est en première forme normale si tout attribut contient une valeur atomique (c'est-à-dire non décomposable et non répétitive).

La normalisation en première forme est une opération triviale mais indispensable comme on va le voir sur les deux exemples que voici :

*) Un attribut est un ensemble de valeurs (répétitif)

Diplômes	
Prof	*Cours*
Albert	Math, Chimie, Bio
Marie	Français, Latin
Marc	Dessin

Diplômes	
Prof	*Cours*
Albert	Math
Albert	Chimie
Albert	Bio
Marie	Français
Marie	Latin
Marc	Dessin

Figure VI-3.

*) Un attribut est lui-même une relation (décomposable) et comporte peut-être des redondances cachées que nous pourrons éliminer ensuite. ("5000 est le code postal de Namur" est dit deux fois par exemple.)

Adresses	
Nom	*Adresse*
Didier	Rue de la Fourche, 45, 5000, Namur
Antoine	Avenue des Roses, 103B, 5100, Jambes
Véronique	Place du 8 Mai, 8, 5000, Namur

Adresses				
Nom	*Rue*	*No*	*C.P.*	*Localité*
Didier	Rue de la Fourche	45	5000	Namur
Antoine	Avenue des Roses	103B	5100	Jambes
Véronique	Place du 8 Mai	8	5000	Namur

Figure VI-4.

b) Deuxième forme normale

La seconde (de même que la troisième) forme normale fait appel à une notion importante dont nous reparlerons souvent : la notion de **dépendance fonctionnelle**. Prenons un exemple :

Elèves			
Nom	*Date Naissance*	*Sexe*	*Classe*

Figure VI-5.

Dans la relation ELEVES, on dira entre autre qu'il existe une dépendance fonctionnelle entre NOM et CLASSE. En effet, la connaissance de NOM détermine celle de la CLASSE. Ce que l'on représente :

NOM -------> CLASSE

Le mot "fonctionnelle" et la représentation proviennent de l'analogie avec les fonctions mathématiques du type x --> y [=f(x)] où la valeur de y est déterminée par celle de x.

La notion de dépendance fonctionnelle nous permet de préciser la notion de clé.

Une clé (possible) d'une relation est une liste d'attributs telle que l'ensemble des autres attributs de la liste est en dépendance fonctionnelle par rapport à l'ensemble des attributs de cette liste et qu'il n'existe pas de sous-ensemble de cette liste qui possède la même propriété.

C'est donc la "plus petite" liste d'attributs qui soit capable de déterminer tous les autres . Nous pouvons à présent définir le seconde forme normale (2NF) :

Une relation est en deuxième forme normale si elle est d'abord en première forme et si tout attribut n'appartenant pas à une clé dépend totalement de la clé (et non d'une partie).

Voyons cela sur un exemple :

Absences				
No Elev	*Date Début*	*Nom*	*Durée*	*Nature*
13	16/03/87	Cécile	2	Maladie
17	16/03/87	Bernard	1	Rés. Fam.
13	19/03/87	Cécile	1	Maladie
25	20/03/87	Anne	1	Maladie

Figure VI-6.

La table VI-6 est bien en première forme normale et la clé primaire qui s'impose comme la plus simple est formée de NOELEV et de DATEDEBUT, car la connaissance de ces deux attributs permet d'identifier clairement l'absence dont on parle. Cette table n'est cependant pas en seconde forme normale, car l'attribut NOM ne dépend fonctionnellement que d'une partie de la clé (à savoir de NOELEV). On voit d'ailleurs dans les quelques lignes données que cette situation introduit une redondance : l'information "l'élève CECILE porte le n° 13" est présente deux fois.

Le passage en seconde forme normale s'obtient en éclatant la table en deux comme présenté à la figure VI-7.

Elèves	
No Elev	*Nom*
13	Cécile
17	Bernard
25	Anne

Absences			
No Elev	*Date Début*	*Durée*	*Nature*
13	16/03/87	2	Maladie
17	16/03/87	1	Rés. Fam.
13	19/03/87	1	Maladie
25	20/03/87	1	Maladie

Figure VI-7.

Remarque : toute relation en 1NF dont la clé et simple est automatiquement en deuxième forme normale puisqu'il n'y a pas dans ce cas de "partie de la clé".

c) Troisième forme normale

Toutes les sources de redondances ne sont pas encore éliminées par la seconde forme. Ainsi, par exemple, on pourra trouver dans une table un attribut qui dépend bien entendu de la clé mais qui est également en dépendance fonctionnelle vis-à-vis d'un autre attribut, non-clé, de la même table.

Considérons la table REPERTOIRE que voici :

Répertoire				
Nom	*Rue*	*No*	*CodePost*	*Localité*

Figure VI-8.

On a les dépendances fonctionnelles suivantes :

NOM -----> RUE
NOM -----> N°
NOM -----> CODE POSTAL
NOM -----> LOCALITE

mais on a aussi la dépendance :

CODE POSTAL -----> LOCALITE

Il s'agit là d'une dépendance transitive car on a bien (et les habitués de math modernes retrouveront un schéma classique) :

Cette dépendance fonctionnelle transitive devra être évitée car elle amène aussi des redondances superflues. Par exemple, à chaque fois que l'attribut CODE POSTAL aura la valeur "5000", on trouvera "NAMUR" dans la colonne LOCALITE. La solution pour éviter cette redondance consiste ici aussi à scinder la table REPERTOIRE en deux nouvelles tables comme ci-dessous :

Répertoire			
Nom	*Rue*	*No*	*CodePost*

Poste	
CodePost	*Localité*

Figure VI-9.

Grâce à la transitivité, la dépendance fonctionnelle NOM -> LOCALITE pourra être facilement reconstruite par jointure des deux nouvelles tables.

On a ainsi la définition de la troisième forme normale (3 NF).

Une relation est en troisième forme normale si elle est d'abord en seconde forme et si tout attribut non-clé ne dépend pas d'un autre attribut non-clé.

Cette définition peut encore être résumée en disant que tous les attributs non-clés sont **mutuellement indépendants**.

La décomposition des tables d'une base jusqu'à ce qu'elles soient toutes en troisième forme normale permet d'éliminer la plupart des problèmes de redondance et est souvent suffisante pour les applications classiques. Néanmoins, quelques anomalies peuvent encore subsister. Les quatrième et cinquième formes normales ainsi que la forme de Boyce-Codd visent à les éliminer.

Nous nous limiterons ici à l'étude de la 4 NF. Le lecteur intéressé trouvera dans la bibliographie présente à la fin de ces notes des titres d'ouvrages (particulièrement ceux de GARDARIN et de MIRANDA et BUSTA) où toutes les règles de normalisations sont présentées en détail.

d) Quatrième forme normale

De nombreux exemples présentés jusqu'ici nous ont permis de mieux comprendre la notion de dépendance fonctionnelle entre deux attributs d'une même relation. On a vu qu'une telle dépendance peut être représentée (si A et B sont des attributs d'une relation et que B dépend fonctionnellement de A) par :

A --------> B

On sait alors que, à toute valeur de A correspond une et une seule valeur de B (par exemple, à toute valeur du NOM correspond une et une seule valeur de la DATE DE NAISSANCE).

Or,si l'on examine de près quelques tableaux, on verra que les attributs ne sont pas toujours liés à la clé par des dépendances fonctionnelles. Reprenons par exemple la table DIPLOMES élaborée précédemment :

Diplômes	
Prof	*Cours*
Albert	Math
Albert	Chimie
Albert	Bio
Marie	Français
Marie	Latin
Marc	Dessin

Figure VI-10.

Il est manifeste qu'il existe une dépendance entre les valeurs des COURS et celles des PROF(S). Il ne s'agit pas d'une dépendance fonctionnelle car à chaque prof ne correspond pas un et un seul cours. Il s'agit ici d'une forme plus générale de dépendance baptisée **dépendance multivaluée**. A une valeur de l'attribut PROF correspondent "de multiples" valeurs de COURS. En soi, une telle dépendance n'est pas problématique. Il n'y a problème que si deux colonnes de la table sont en dépendance multivaluée par rapport à une troisième et que ces deux colonnes sont mutuellement indépendantes.

Voyons cela par un exemple (tiré de GARDARIN) :

Etudiant		
NoElev	*Cours*	*Sport*
100	Informatique	Tennis
100	Informatique	Football
200	Informatique	Vélo
200	Mathématique	Vélo

Figure VI-11.

Cette table est bien en troisième forme normale puisque tous les attributs sont nécessaires pour former la clé. On y trouve cependant des redondances manifestes (le fait que Michel suit le cours d'info est dit deux fois par exemple).

Aussi, pour éliminer cette redondance, nous éclaterons aussi cette table de la façon suivante :

Etu-Cours	
NoElev	*Cours*
100	Informatique
200	Informatique
200	Mathématique

Etu-Sport	
NoElev	*Sport*
100	Tennis
100	Football
200	Vélo

Figure VI-12.

Nous laisserons au spécialiste le soin de définir (clairement?!) la quatrième forme normale. A notre niveau, nous nous contenterons de retenir que cette forme normale vise à éliminer au maximum les dépendances multivaluées des relations. En principe, on n'en conservera qu'une seule par table.

Nous nous arrêterons donc ici dans l'énoncé des règles de normalisation des tables. Rappelons que le but de cette normalisation est de donner des "outils" au concepteur de la base de données pour qu'il puisse choisir une répartition des données qui soit efficace avec un SGBD relationnel et ce en éliminant les redondances d'informations.

VII. Structure d'une base de données : trois niveaux

Notre découverte des bases de données nous à conduit à faire connaissance avec une belle quantité de concepts et de mots nouveaux. Nous avons notamment vu quels outils nous sont offerts par les SGBD pour créer et manipuler une base de données.

Il faut cependant bien se rappeler que ces outils ne nous permettent que de **mettre en oeuvre** un **schéma** de base de données qui aura été imaginé auparavant par l'homme (en général le concepteur de la base de données). Le bon fonctionnement et la facilité d'utilisation de la base dépendent donc crucialement de la qualité du travail de **conception** de la BD. Nous avons déjà vu un des outils disponibles pour aider le concepteur dans sa tâche : ce sont les règles de normalisation. Nous en évoquons d'autres dans le chapitre suivant.

Divers organismes se sont penchés sur le problème de la conception des bases de données. Ce sont entre autres CODASY (Conference On Data Systeme Language), ISO (International Standard Organisation) et ANSI (American National Standard Institute) qui ont proposé une organisation de BD suivant une architecture à trois niveaux (ou schémas) : le schéma conceptuel, les schémas externes et le schéma interne.

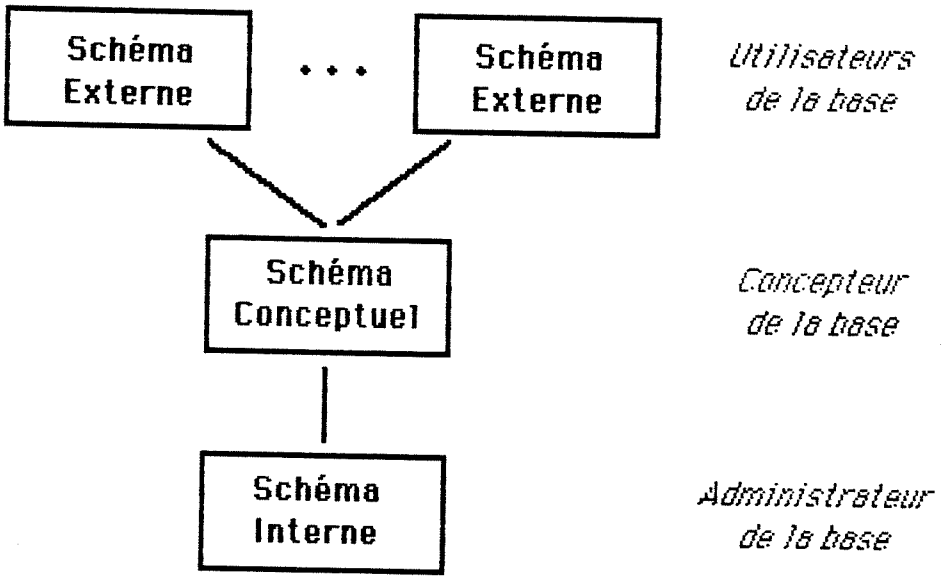


fig VII-1 Architecture d'une base de données

Cette découpe en trois niveaux a pour objectif essentiel d'assurer l'indépendance entre l'organisation logique et l'implémentation physique des informations.

3. Les schémas externes

L'utilisation d'une base de données par plusieurs utilisateurs fait apparaître des besoins spécifiques pour chacun d'eux.

Dans le cas d'une école, l'économat n'a pas besoin des mêmes informations que le service de mise au point des horaires par exemple. Chaque utilisateur n'a donc pas besoin de pouvoir accéder à toutes les informations de la base de données. Il est même intéressant pour des raisons de sécurité et de confidentialité de créer pour chaque utilisateur une **vue** de la base qui lui présente les données *dont il a besoin* et de la façon *la plus claire pour lui*.

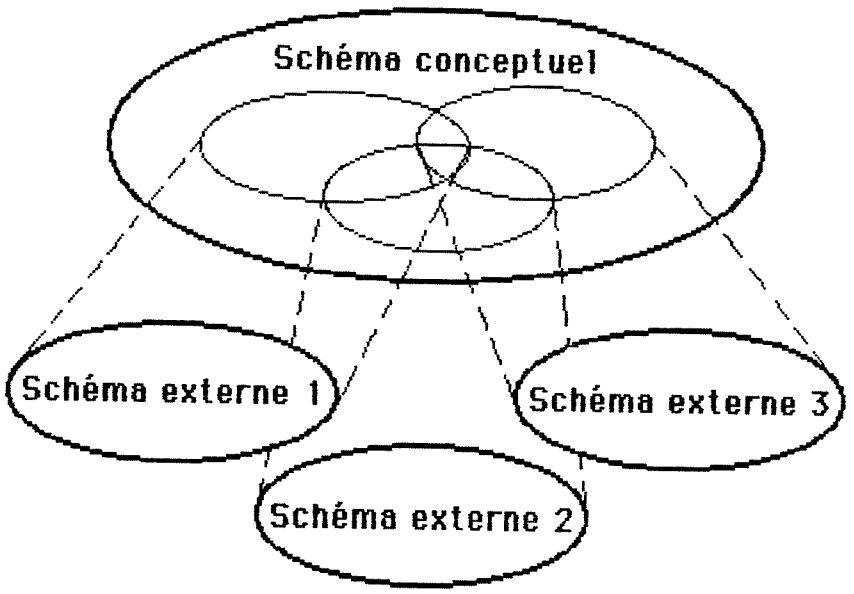


Figure VII-2.
Schémas externes dérivés de sous-ensembles du schéma conceptuel

Il ne lui sera, par contre, pas possible d'accéder au reste des informations de la BD.

Les schémas externes ne sont rien d'autre que ces vues (*) adaptées aux utilisateurs ou aux applications utilisables pour des (groupes d') utilisateurs.

Dans le cas des BD relationnelles, les vues seront souvent bien plus faciles à employer que les tables de bases car elles correspondront généralement à la représentation de fichiers classiques avec par exemple la redondance (visuelle !) de certaines informations.

Nous verrons que dBASE III dans sa version "plus" offre aussi la possibilité de créer assez simplement des vues. Il ne s'agit cependant que d'une combinaison astucieuse de possibilités déjà présentes dans la

version de base et qui en conserve par conséquent les limitations.

(*) Le terme "vue " est d'ailleurs repris par la terminologie des BD relationnelles pour désigner les schémas externes.