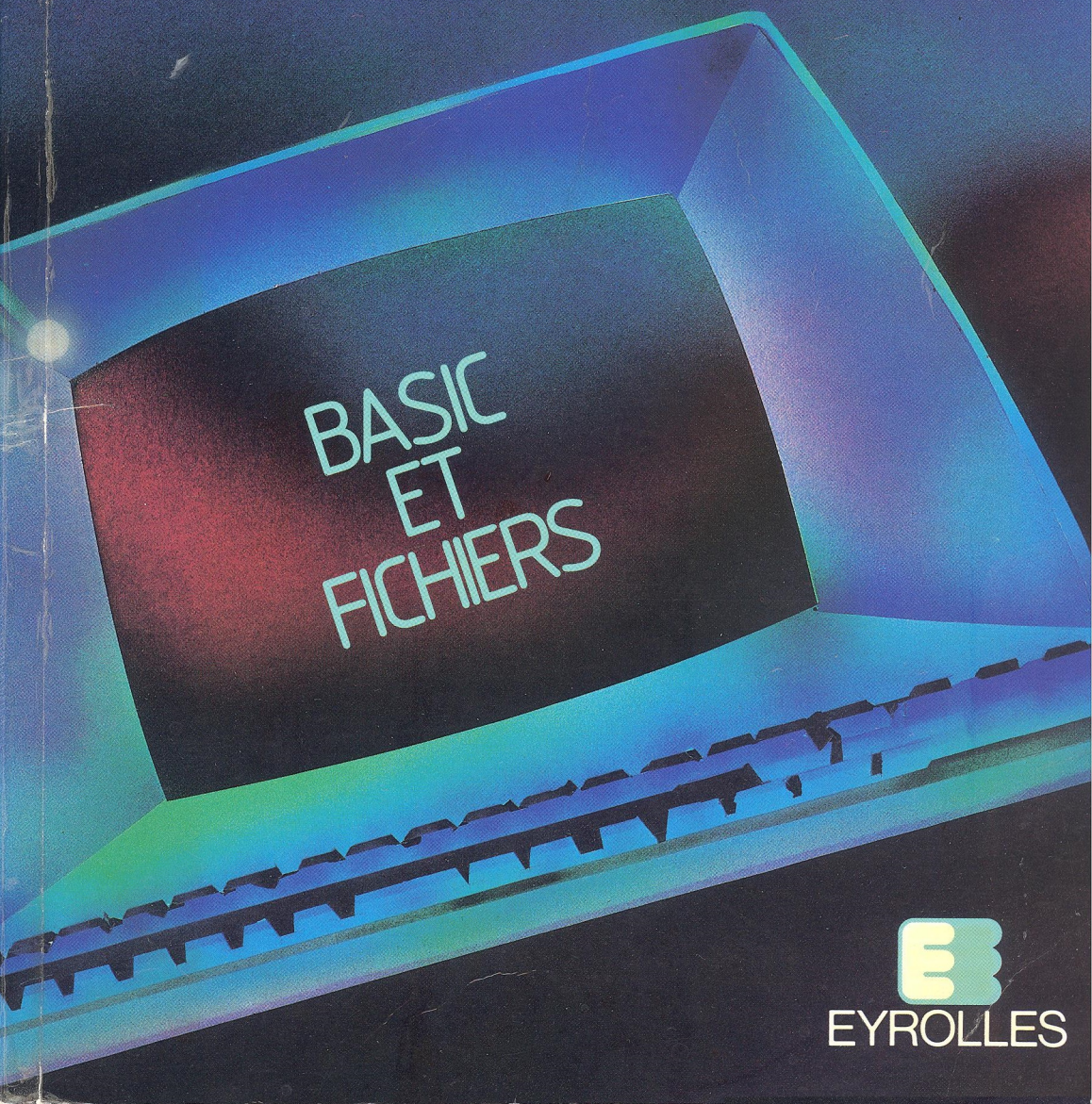
PROGRAMMATION SUR AMSTRAD PCW 8256/8512

PATRICE BIHAN



EYROLLES

PROGRAMMATION SUR AMSTRAD PCW 8256/8512 : Basic et Fichiers

CHEZ LE MEME EDITEUR

DELANNOY - *Je débute en basic AMSTRAD.*

MILSANT - *Lexique d'informatique et de micro-informatique avec index alphabétique anglais - français.*

DELANNOY - *Initiation à la programmation.*

SCHMIT - *Eléments de programmation.*

de ROSSI - *Apprentissage rapide du BASIC.*

DELANNOY - *Apprendre à programmer en BASIC.*

LEPAPE - *L'assembleur facile du Z.80.*

AUBERT, SCHOMBERG - *Pratiquez l'intelligence artificielle.*

STUMM - *Jeux de robotique en BASIC.*

DELAHAYE - *Dessins géométriques et artistiques avec votre micro-ordinateur.*

DELAHAYE - *Nouveaux dessins géométriques et artistiques avec votre micro-ordinateur.*

PEZERET - *Créez vos jeux d'aventure sur micro-ordinateur. Méthodes et idées.*

MANCHON, NASR - *Physique sur micro-ordinateur.*

PROGRAMMATION SUR AMSTRAD PCW 8256/8512 : Basic et Fichiers

par
Patrice BIHAN

EYROLLES
61, boulevard Saint-Germain – 75005 PARIS
1986

Si vous désirez être tenu au courant de nos publications, il vous suffit d'adresser votre carte de visite au :

Service « Presse », Editions EYROLLES,
61, Boulevard Saint-Germain
75240 PARIS CEDEX 05,

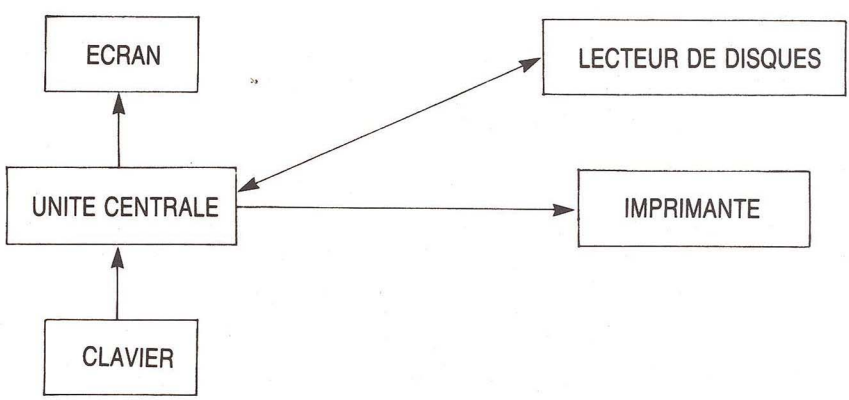
en précisant les domaines qui vous intéressent. Vous recevrez régulièrement un avis de parution des nouveautés en vente chez votre libraire habituel.

TABLE DES MATIERES

INTRODUCTION

L'ordinateur est un merveilleux outil d'analyse de données et de stockage d'information. Il est aussi plus simplement source de plaisir intellectuel.

Cependant, il convient de connaître au préalable son environnement, c'est-à-dire l'ensemble du système informatique, et, pour ce qui nous concerne, l'AMSTRAD PCW 8256/8512.



L'unité centrale est environnée d'objets précisément nommés "périphériques". Ces derniers sont utilisables grâce aux ordres venus de l'unité centrale qui les supervisent. Ces ordres sont rendus possibles grâce à des instructions (le programme) stockés en mémoire dans l'ordinateur. La mémoire se divise en deux parties : la RAM (de ran-

dom acces memory, mémoire à accès aléatoire, ou bien encore mémoire vive, MEV) et la ROM (de read only memory, mémoire accessible seulement en lecture, ou mémoire morte, MEM). Pour ces deux types de mémoire, on peut dire très grossièrement que la RAM contiendra le programme et la ROM contiendra les codes et les instructions nécessaires au bon fonctionnement du programme. Le programmeur peut créer des programmes (c'est la moindre des choses !) et donc entrer dans la RAM. Par contre, il n'a pas accès à la ROM, qui contient des informations entrées définitivement par le concepteur et inaltérables, d'où le nom de "mémoires morte". Ces mémoires sont constituées d'emplacements nommés "octets", qui représentent 8 chiffres binaires. On peut donc mesurer l'importance de la mémoire d'un ordinateur au nombre d'octets : ainsi, un ordinateur ayant 30 000 octets aura une mémoire vive plus grande qu'un ordinateur ayant 10 000 octets. Pour ce qui nous concerne, la quantité de mémoire disponible avoisine en BASIC 30 000 octets : c'est largement suffisant. Pour simplifier, on compte les octets en Kilo-octets : 30 000 octets = 30 K-Octets.

Ainsi, programmer en BASIC revient à entrer dans la mémoire vive une série d'instructions simples qui, une fois traduite selon un code que l'ordinateur connaît, se retrouve intégrée dans la RAM sous forme d'octets directement analysables par le système. Ce que nous allons apprendre dans cet ouvrage correspond à cette série d'instructions, nommée "programme".

Présentation de l'ouvrage :

Il est conçu comme un cours progressif en pensant particulièrement au programmeur débutant et destiné à éviter à ce dernier des erreurs et des tatonnements douloureux ; l'ouvrage ne couvre donc pas toutes les possibilités de l'AMSTRAD mais va directement à l'essentiel. Cependant, le programmeur confirmé y trouvera certainement matière à réflexion avec le dernier chapitre concernant les fichiers, notamment les fichiers à accès indexé qui utilisent JETSAM, le gestionnaire intégré de fichiers.

Conventions d'écriture :

Les instructions BASIC sont toujours indiquées en majuscules : c'est simplement pour mieux les faire ressortir au lecteur : on peut bien sûr les écrire en minuscules. Ceci est également vrai pour le nom des valeurs.

1.
PRÉSENTATION INFORMELLE

1.1. PREMIERS CONTACTS — LE MODE DIRECT

Après avoir introduit la disquette CP/M Plus, le signe A> s'affiche sur l'écran quelques secondes plus tard. Il signifie simplement que le lecteur de disquette est prêt à fonctionner : l'ordinateur est placé sous le contrôle du système d'exploitation de disques appelé CP/M qui nomme précisément le lecteur par la lettre A. Un second lecteur s'appellerait B, un troisième C etc. Votre AMSTRAD, en version de base, ne possède qu'un seul lecteur (Nous verrons ultérieurement qu'en réalité il existe une mémoire interne qui peut faire office de second lecteur, appelé M).

Pour utiliser le Basic, il faut placer ce dernier, stocké sur la disquette, en mémoire dans l'ordinateur. Cela se fait simplement en entrant BASIC suivi de RETURN :

A>BASIC + touche RETURN

Le Basic affiche alors son message de bienvenue puis un OK. Vous allez le retrouver souvent : c'est ce message qui apparaîtra à chaque fois que l'ordinateur aura suivi une commande ou exécuté un programme. Son absence est significative : l'ordinateur est ou bien en train de faire un long calcul, ou alors complètement planté (cela arrive quand même très rarement).

Sans plus tarder, commençons notre apprentissage.

Comme l'écran affiche des messages, nous pouvons également

le faire avec l'instruction PRINT (imprimer en anglais : vieille résurgence des débuts de l'informatique, quand l'imprimante existait avant la télévision). Pour afficher le nombre 123, il suffit de taper :

PRINT 123. Ce faisant, il ne se passe rien ; en effet il est nécessaire de valider l'ordre donné à l'ordinateur avec la touche RETURN (de retour chariot : comme sur les machines à écrire). Cette touche est certainement la plus utilisée en informatique : à chaque fois que l'on désire entrer dans l'ordinateur une quelconque information, il faut faire enregistrer la commande par RETURN, ainsi :

PRINT 123 RETURN

REMARQUES : — il faut frapper la touche RETURN et non entrer les lettres R.E.T.U.R.N.
— une autre touche a le même usage : ENTER.
— il est indifférent d'écrire les instructions Basic en majuscules ou en minuscules.

Après avoir entré RETURN, le nombre 123 s'affiche bien sur l'écran, et de nouveau OK apparaît.

Nous pouvons ainsi écrire des nombres sur l'écran. Nous pouvons également faire des opérations et utiliser l'ordinateur comme une calculatrice de la même manière :

— Pour additionner les nombres 123, 34, 45 :

PRINT 123 + 34 + 45 suivi de RETURN
ou PRINT 123+34+45 suivi de RETURN

— Pour diviser des nombres :

PRINT 124 / 45 suivi de RETURN (et non :)

— Pour multiplier des nombres :

PRINT 34 * 45 suivi de RETURN

(Le signe de la multiplication se trouve sur la touche < : c'est une astérisque pour ne pas le confondre avec un x minuscule)

— Pour soustraire des nombres :

PRINT 45 – 6 suivi de RETURN

— Pour élever à une puissance

PRINT 4 ↑ 3 suivi de RETURN

En utilisant PRINT de cette manière, on demande deux choses simultanément :

1) Faire un calcul
2) Afficher le résultat

1.2. LE MODE PROGRAMME

1.2.1. Numérotation des lignes

Nous venons d'utiliser l'ordinateur en mode direct. C'est-à-dire que nous n'avons pas encore appris à programmer ; pour ce faire, il faut apprendre une notion simple : pour écrire un programme, il faut dire à l'ordinateur que l'on désire passer du mode direct au mode programmation. Cela se fait tout simplement en écrivant un numéro devant l'instruction Basic, par exemple :

1 PRINT 45 + 67 suivi DE RETURN

Pourquoi ce numéro ?

Un programme Basic est une succession d'instructions, de type PRINT ou autres, et ces instructions se placent dans un ordre précis, assigné par le programmeur, et donc numérotées. En fait, on numérote les lignes du programme où se trouvent ces instructions. On pourrait donc écrire les opérations précédentes de la manière suivante :

```
1 PRINT 123 + 34 + 45 suivi de RETURN
2 PRINT 124 / 45 suivi de RETURN
3 PRINT 34 * 45 suivi de RETURN
4 PRINT 45 - 6 suivi de RETURN
5 PRINT 4↑3  suivi de RETURN
```

Vous avez remarqué que, contrairement au mode direct, l'ordinateur se contente d'enregistrer les lignes une par une, sans pour autant afficher les résultats ; cela est dû au fait que vous ne lui avez pas donné l'ordre d'afficher ces résultats ! Vous avez simplement demandé d'enregistrer, pas de calculer, ni d'afficher.

On peut (quand même...) voir le résultat des opérations, en demandant à l'ordinateur d'exécuter le programme avec l'ordre :

RUN (de run : courir, démarrer) suivi de RETURN.

Aussitôt, les résultats s'affichent, et OK aussi.

IMPORTANT :

Comme la touche RETURN est omniprésente, il n'en sera plus fait mention à la suite : il ne faudra pas cependant l'oublier !

Quelle est la différence entre le mode direct et le mode programme, dans les exemples précédents ?

Quand on entre PRINT 123

et 1 PRINT 123, il n'y a aucune différence quant au résultat. Il se trouve qu'en mode direct, une fois l'instruction exécutée, l'ordinateur oublie cette instruction et attend l'instruction suivante, alors qu'en mode programme, l'instruction (ou la série d'instructions) est toujours présente : et si elle est toujours présente, on peut donc la faire exécuter autant de fois que l'on veut et ceci sans la réécrire, simplement grâce à l'ordre RUN. Par extension, on peut également se dire que, s'il existe un moyen de sauvegarder le programme, au cas où on éteindrait l'ordinateur, on pourrait donc s'en resservir un autre jour ! C'est exactement ce qu'il est possible de faire avec l'AMSTRAD 8256 (soyons honnêtes : c'est bien sûr possible avec tous les ordinateurs).

1.2.2. Point virgule, virgule, point

Vous avez remarqué certainement que les résultats s'affichent les uns derrière les autres. En d'autres termes, chaque ordre PRINT suivi d'un argument quelconque provoque, outre l'écriture, le passage automatique à la ligne : il en sera toujours ainsi, sauf spécification contraire de votre part. Par exemple, vouloir afficher sur l'écran les trois nombres suivants 8, 16 et 25 sur la même ligne se fera en utilisant ou bien une virgule, ou bien un point-virgule, ou bien rien (!) de la manière suivante :

PRINT 8,16,25 → écrit les trois valeurs avec un grand espace entre elles 8 16 25

mais

PRINT 8;16;25 → écrit les trois valeurs séparées par un petit espace 8 16 25.

Il faut savoir que l'écran de l'ordinateur est constitué de 90 colonnes. Ecrire les trois nombres avec la virgule oblige à une tabulation automatique de 16 en 16 colonnes, d'où l'espace entre ces nombres ; par contre, le point-virgule entraîne l'écriture de la valeur suivante juste à côté de la précédente. Il n'est donc pas nécessaire d'écrire autant de PRINT que de valeurs à afficher. Il faut cependant bien comprendre que :

```
1 PRINT 1,2,3
2 PRINT 4,5,6
```

affichera :

```
1   2   3
4   5   6
```

Par contre :

```
1 PRINT 1;2;3
2 PRINT 4;5;6;
```

affichera :

```
 1  2  3
 4  5  6
```

et :

```
1 PRINT 1,2,3,
2 PRINT 4,5,6
```

affichera

```
 1      2      3      4      5      6
```

La différence tient dans la dernière virgule de la ligne n° 1 : elle oblige à afficher les valeurs suivantes à la suite des valeurs précédentes ; de même :

```
1 PRINT 1;2;3;
2 PRINT 4;5;6
```

affichera

```
 1  2  3  4  5  6
```

Conseil : Entraînez-vous à faire plusieurs manipulations et amusez vous à mélanger les virgules et les points-virgules comme ceci :

```
PRINT 1 , 2;3;4,,,,5,6
```

N'oubliez surtout pas que la ponctuation en BASIC n'a rien à voir avec une ponctuation normale ; outre le point-virgule et la virgule que nous venons de voir, le point sert simplement de virgule décimale (comme sur les calculatrices, à la mode anglo-saxonne) et les deux points verticaux (:) ont une autre fonction Basic que nous verrons ultérieurement. Il ne faut donc pas utiliser cette ponctuation pour écrire des programmes.

1.2.3. Le premier programme

Posons-nous le problème suivant :
Nous sommes désireux de calculer un prix TTC. On connaît la valeur de la TVA, 18.6 %, ainsi que le prix Hors-Taxes, 120 francs. La formule permettant de trouver le résultat est simple :

Prix TTC = Prix Hors-taxes + (Prix hors-taxes * TVA/100)

Ainsi :

Prix TTC = 120 + (120 * 18.6 / 100) = 142.32

Trois notions importantes apparaissent ici :

— Pour transposer ce problème en Basic, il faut toujours écrire une formule de calcul de cette manière, c'est-à-dire le nom du résultat AVANT le signe " = " et non après, comme on le fait habituellement (et fort logiquement d'ailleurs) en arithmétique, où l'on aurait écrit de préférence : Prix hors-taxes + (Prix hors-taxes * TVA / 100) = Prix TTC. La raison en est que le Basic analyse d'abord ce qu'il y a à droite du signe " = ", calcule et range le résultat sous le nom écrit à gauche du signe.

— Dans cet exemple, les valeurs servant au calcul portent un nom : Prix TTC, Prix Hors-taxes ou TVA. En Basic, il est obligatoire d'écrire ces noms en UN mot comme ceci : Prix TTC, ou PHT par exemple.

— Les signes tels le pourcentage (%) ou les mots accessoires, comme "francs" ne doivent pas être écrits dans une formule de calcul. Ils provoqueraient un message d'erreur ou un blocage du système ou encore un résultat inattendu.

Ceci dit, nous pouvons maintenant écrire notre programme de la manière suivante :

```
1  TVA = 18.6
2  PHT = 120
3  PTTC= PHT + ( PHT * TVA / 100 )
4  PRINT PTTC
```

Puis lançons l'exécution du programme en faisant RUN :

l'écran affiche le bon résultat : 142.32.

Ce programme appelle plusieurs commentaires, sur la forme et le fond.

Bien qu'il fournisse le bon résultat, il fournira toujours le même résultat à chaque fois que nous ferons RUN. Il serait éminemment souhaitable de pouvoir calculer une TVA sur d'autres valeurs. Nous pouvons en fait corriger la ligne 2, celle où se trouve la valeur qui varie, PHT.

Pour corriger, il suffit d'éditer la ligne, c'est-à-dire de la sortir du programme en faisant simplement :

EDIT 2

Aussitôt, la ligne n° 2 s'affiche ; il suffit alors d'appuyer sur la flèche (→) autant de fois qu'il le faut pour amener le curseur sur le 1 de 120, puis effacer avec la touche DEL ← les trois chiffres de 120 et les remplacer par d'autres, 234 par exemple, et terminer la correction en frappant, bien sûr, RETURN.

Pour juger de la correction effective du programme, rien ne vaut une vérification. Elle se fait avec l'ordre LIST qui, comme son nom l'indique, liste le programme :

```
LIST
1 TVA =18.6
2 PHT = 234
3 PTTC = PHT + ( PHT * TVA / 100 )
4 PRINT PTTC
```

Si, de nouveau, nous lançons RUN, le programme va calculer à partir de la nouvelle valeur de PHT et trouver 277.524.

Arrivés à ce stade de l'analyse, l'auteur de cet ouvrage peut bien vous avouer qu'il vous a honteusement menti ! En réalité, et bien que cela ait parfaitement fonctionné, nous venons de le voir, la commande EDIT ne sert pas précisément à changer les valeurs dans un programme ; cela serait, de cette manière, particulièrement pénible, surtout dans le cas où l'on désirerait calculer une TVA sur des centaines de Prix hors-taxes. Nous venons simplement d'apprendre, et vous l'aviez compris, comment corriger une ligne Basic au cas (relativement courant, disons-le...) où une erreur se serait glissée. Cette erreur peut prendre plusieurs formes : oubli d'un argument, d'un signe, mot Basic mal orthographié etc. Lors de l'exécution du programme par la commande RUN, le Basic l'analyse ligne après ligne ; si une erreur de ce type est détectée, il vous tient au courant avec le message :

SYNTAX ERROR, qui, comme chacun sait, signifie ERREUR de SYNTAXE. Il ne vous reste plus, alors, qu'à éditer la ligne erronée et chercher puis corriger l'erreur ; il va sans dire qu'une ligne peut comporter plusieurs erreurs : le système s'arrêtera alors toujours sur la même ligne, jusqu'à la correction de la dernière erreur.

Pendant que l'on se trouve à parler de l'éditeur, notez quelques petites astuces pour éviter trop de manipulations fastidieuses par la suite. Supposons la ligne 3 écrite ainsi :

3 PTTC = PHT + (PHT * TVA/100

Une erreur de syntaxe transparait : il manque une parenthèse fermante. Pour corriger cette ligne, il suffit de faire EDIT 3 (RETURN), puis d'emmener le curseur jusqu'à l'erreur, soit en fin de ligne. On peut, pour éviter cela, faire juste après EDIT 3, ceci :

2 fois touche FIND 0. Qu'est-ce à dire ?

Simplement, on a demandé à l'éditeur (le système qui permet de corriger) de positionner le curseur sur le deuxième caractère zéro de

la ligne ; ce faisant, la ligne s'écrit immédiatement sous vos yeux, sans qu'il soit besoin de frapper la flèche (→). Notez qu'on aurait pu faire encore plus simple : pour aller directement à la fin, il suffisait de frapper deux fois de suite la touche FIND.

Si vous venez de valider une ligne avec RETURN et que, pour une raison ou pour une autre, vous désirez réentrer dans l'éditeur, il suffit de frapper immédiatement après RETURN, ALT et A, simultanément. Ceci aura pour effet de réécrire la ligne, pour pouvoir éventuellement la corriger de nouveau.

Pour effacer des caractères à droite du curseur, il suffit de frapper deux fois de suite la touche CUT.

Pour effacer définitivement une ligne de programme, il suffit d'entrer son numéro et RETURN.

Pour effacer un groupe de lignes, on entre :

DELETE n° première ligne - n° dernière ligne (de delete, effacer).

Sortons maintenant de l'éditeur et revenons à la question précédemment posée : comment entrer différents prix hors-taxes sans pour autant utiliser la commande EDIT ?

Une instruction Basic va servir à cela. Elle s'appelle fort justement INPUT (pour entrer). C'est une commande qui permet de communiquer avec l'ordinateur. En effet, quand un programme en train de s'exécuter rencontre cette instruction, il s'arrête car il attend une entrée de l'utilisateur.

Dans notre exemple, on désire entrer une valeur différente dans PHT ; notre ligne n° 2 sera écrite ainsi ;
2 INPUT PHT

Entrons cette ligne et listons le programme :

```
   LIST

1 TVA = 18.6
2 INPUT PHT
3 PTTC = PHT + ( PHT * TVA / 100 )
4 PRINT PTTC
```

Une autre caractéristique de la correction apparaît : le simple fait d'écrire une ligne dont le numéro est le même qu'une autre remplace automatiquement cette dernière.

Faisons de nouveau RUN ; l'écran affiche un point d'interrogation qui signifie toujours que l'ordre INPUT vient d'être rencontré. Tant

que l'utilisateur du programme n'entre rien, il ne se passe strictement rien. Pour continuer, il faut entrer une valeur quelconque ; sitôt cette dernière entrée, l'ordinateur peut continuer l'exécution de son programme, puisqu'il connaît maintenant la valeur de PHT :

```
RUN
?
345 (RETURN)
409.17
Ok
```

Maintenant, il est beaucoup plus facile d'utiliser l'ordinateur ; il suffit de faire RUN, puis d'entrer simplement la valeur du prix hors-taxes. Les manipulations sont plus aisées, car, pour un calcul on doit entrer :

```
RUN
la valeur
touche RETURN
```

Il faut noter que INPUT s'utilise TOUJOURS suivi d'un argument qui représente la valeur.

On peut cependant faire encore plus simple en intégrant l'ordre RUN en ligne n° 5 :

```
1  TVA = 18.6
2  INPUT PHT
3  PTTC= PHT + ( PHT * TVA / 100 )
4  PRINT PTTC
5  RUN
```

Maintenant, un premier RUN suffit, pour démarrer le programme. Quand le système qui analyse le programme trouve un premier résultat, il l'affiche puis, en rencontrant l'ordre RUN, REPART au début, puisque cet ordre signifie, non seulement de démarrer un programme, mais de le démarrer au début. Ainsi, le programme tourne maintenant sans fin, car rien ne l'arrête, sauf la frappe simultanée de la touche ALT et la lettre C, qui permet le retour au mode programme normal, ou STOP.

Notre programme a un inconvénient relativement gênant, bien qu'il fonctionne : nous savons que le nombre à entrer, lors de l'exécution

du programme représente le prix hors-taxes ; nous le savons parce que c'est nous qui avons conçu le programme. Il serait souhaitable que le programme demande clairement ce prix hors-taxes, avec une phrase du genre « entrez le prix hors-taxes SVP ». Cela est parfaitement possible, avec l'ordre PRINT, bien sûr. A première vue, on pourrait penser que :

PRINT entrez le prix hors-taxes SVP

suffirait. Mais le Basic ne comprendrait pas. Il faut mettre la phrase entre guillemets, pour que le Basic fasse la différence entre une phrase, composée de caractères, et le nom d'une valeur numérique, composée elle aussi de caractères, ainsi :

PRINT ''entrez le prix hors-taxes SVP'' est correcte. Quelle est la différence fondamentale entre cette formulation et PRINT PTTC par exemple. Dans le premier cas, le Basic comprend qu'il doit afficher des caractères ; dans le second, il comprend qu'il doit afficher la VALEUR correspondant en mémoire à ces caractères. Si on omet les guillemets dans le premier cas, le Basic croit que les mots ''entrez, ''le'', ''prix'' etc symbolisent chacun une valeur, tout comme le font les mots TVA, PHT, PTTC.

Ainsi, nous pouvons réécrire le programme de telle manière qu'à l'exécution, il pose une question à l'utilisateur et qu'il affiche ensuite le résultat clairement exprimé :

```
1 TVA =18.6
2 PRINT "entrez le prix hors-taxes SVP "
3 INPUT PHT
4 PTTC = PHT + ( PHT * TVA / 100 )
5 PRINT  " PTTC = "
6 PRINT PTTC
7 RUN
```

Lançons le programme :

```
RUN
 entrez le prix hors-taxes SVP "
 ?
 230
 PTTC =
 272.78
 entrez le prix hors-taxes SVP
 ?
```

```
100
PTTC =
118.16
entrez le prix hors-taxes SVP
touche STOP
ok
```

La réécriture du programme appelle plusieurs remarques.

Plutôt que de réécrire entièrement le programme, il aurait été préférable d'ajouter simplement les lignes nouvelles ; or, il est impossible de le faire, car on ne peut manifestement insérer une ligne entre le numéro 1 et 2, par exemple. Par contre, si le programme initial avait été écrit avec des numéros de ligne allant de 10 en 10 comme ceci :

```
10 TVA = 18.6
20 INPUT PHT
30 PTTC = PHT + ( PHT * TVA / 100 )
40 PRINT PTTC
50 RUN
```

On pourrait facilement intégrer maintenant de nouvelles lignes, en se servant des numéros de ligne non-utilisés :

```
10 TVA = 18.6
15 PRINT " entrez le prix hors-taxes SVP "
20 INPUT PHT
30 PTTC = PHT + ( PHT * TVA / 100 )
35 PRINT " PTTC = "
40 PRINT PTTC
50 RUN
```

On voit que les lignes n° 15 et 35 peuvent facilement être insérées. Il n'est donc plus nécessaire de réécrire TOUT le programme, mais seulement de prendre la précaution au départ d'écrire les lignes en les numérotant de 10 en 10 (ou de 14 en 14, ou bien de 50 en 50 : comme vous voulez). C'est ce que nous ferons maintenant ; remarquez qu'il existe une commande pour les imprévoyants, RENUM. Cette fonction, qui signifie renuméroter sert exactement à ce que nous venons de dire ; pour transformer le programme, il suffisait de faire RENUM puis RETURN, bien sûr, et le programme se retrouve avec la nouvelle numérotation. Si le besoin s'en fait sentir, on peut renommer les lignes avec un écart différent de 10, et à partir de la ligne voulue :

RENUM 500 : tout le programme commence à 500, puis 510, etc ;

```
500 TVA = 18.6
510 PRINT " entrez le prix hors-taxes SVP "
520 INPUT PHT
530 PTTC = PHT + ( PHT * TVA / 100 )
540 PRINT " PTTC = "
550 PRINT PTTC
560 RUN
```

RENUM 1000,530 : le programme est renuméroté à 1000 à partir de la ligne 530.

```
500 TVA = 18.6
510 PRINT " entrez le prix hors-taxes SVP "
520 INPUT PHT
1000 PTTC = PHT + ( PHT * TVA / 100 )
1010 PRINT " PTTC = "
1020 PRINT PTTC
1030 RUN
```

RENUM 1000,500,12 : le programme est renuméroté comme précédemment, mais de 12 en 12.

```
1000 TVA = 18.6
1012 PRINT " entrez le prix hors-taxes SVP "
1024 INPUT PHT
1036 PTTC = PHT + ( PHT * TVA / 100 )
1048 PRINT " PTTC = "
1060 PRINT PTTC
1072 RUN
```

Par ailleurs, on peut remarquer qu'il était plus simple, et plus élégant d'écrire :

PRINT ''prix TTC = '' ; PTTC ou mieux ''Prix TTC = '' ; PTTC ; ''francs'' plutôt qu'utiliser deux lignes. Le point-virgule sert à relier sur une même ligne deux ou plusieurs données à afficher à l'écran (ou, comme nous le verrons bientôt, à l'imprimante).

On peut noter également une autre manière d'écrire avec INPUT ; au lieu de dire :

PRINT ''entrez le prix hors-taxes SVP'' puis
INPUT PHT, on peut plus simplement écrire :
INPUT ''entrez le prix hors taxes SVP'' ; PHT. De cette façon,

l'instruction INPUT permet et d'écrire un texte et d'entrer une valeur. C'est d'ailleurs de cette manière que l'on rencontre le plus souvent cette instruction. Revoyons donc une dernière fois notre programme :

```
10 TVA = 18.6
20 INPUT  " entrez le prix hors-taxes SVP " ; PHT
30 PTTC = PHT + ( PHT * TVA / 100 )
40 PRINT " PTTC = ";PTTC ; "francs"
50 RUN
   Exécutons le programme:
   RUN
   entrez le prix hors-taxes SVP ?
   100
   PTTC = 118.6 francs
   entrez le prix hors-taxes SVP ?
   TOUCHE STOP
   Ok
```

DELETE

On peut effacer une partie d'un programme en spécifiant la ligne commençant cette partie et la ligne la finissant avec DELETE (de delete, effacer) :

DELETE 20-50 effacera dans le programme précédent les lignes 20, 30, 40 et 50.

DELETE –50 effacera TOUTES les lignes du début jusqu'à 50.

DELETE 30– effacera TOUTES les lignes de la 30 à la fin du programme.

NEW

On peut également avoir besoin d'effacer tout le programme pour pouvoir en écrire un autre ; on se servira alors de NEW (de new, nouveau) qui supprime TOUT le programme.

1.3. TOUCHES SPÉCIFIQUES DE L'AMSTRAD PCW 8256/8512 :

STOP : arrête l'exécution du programme. Non ? Si ! Etonnant !

ALT et C (simultanément) : idem

ALT et S (^ ^) : arrête provisoirement l'exécution. La frappe d'une touche quelconque (sauf STOP) redémarre l'exécution.

CUT : Deux pressions successives éliminent tout ce qui se trouve à droite du curseur dans une ligne.

FIND : Recherche d'un caractère spécifié dans une ligne et positionne automatiquement le curseur dessus :

3 fois FIND S, par exemple, positionne le curseur sur la troisième occurrence (apparition) de la lettre S.

FIND deux fois positionne le curseur directement à la fin de la ligne.

ALT et A : réécrit la dernière instruction ou ligne entrée.

Exercices divers sur le chapitre précédent

Dans la mesure du possible, essayez de deviner les réponses puis essayez après sur l'ordinateur.

A chacun des petits programmes suivants, que se passe-t-il si nous faisons RUN ?

Ex n° 1 :

```
10 RUN
```

Ex n° 2 :

```
10 PRINT 10
20 PRINT
43 PRINT 43
50 PRINT  "c'est fini ! "
```

Ex n° 3 :

```
100 PRINT " entrez la longueur de votre table "
200 INPUT "entrez la longueur de votre table ";long
300 INPUT "entrez la largeur de votre table ";larg
400 PRINT " la surface de votre table est de "; LONG *
LARG
```

Vous entrez 120 pour la longueur et 156 pour la largeur

Ex n° 4 :

```
10 PRINT
```

```
20 PRINT
30 PRINT
```

Ex n° 5 :

```
10 PRINTTVA
20 PRINTPTTC
```

Solutions

Ex. n° 1 :

Il ne se passe apparemment rien, mais ok ne s'affiche pas. L'ordinateur rencontre un RUN qui l'oblige à repartir au début du programme ; or ce début, c'est RUN (!), ce qui l'oblige de nouveau à repartir, et ceci sans jamais s'arrêter. C'est donc un programme parfaitement inutile... Mais notez que de nombreux débutants incluent souvent accidentellement un RUN dans un programme : nous l'avions fait dans les exemples précédents, mais seulement à la fin du programme, jamais au début.

Ex. n° 2 :

```
RUN
10

43
```

c'est fini !

Le print en 20 sert seulement à passer une ligne.

Ex. n° 3 :

```
RUN
entrez la longueur de votre table
entrez la longueur de votre table
120
entrez la largeur de votre table
156
La surface de votre table est de 18720
```

Les lignes 100 et 200 se répètent à l'affichage, mais ne gênent pas le déroulement du programme. Le nom des valeurs s'écrit indifféremment en majuscules ou en minuscules sans problème. Vous avez sans doute remarqué dans cet exemple la longueur curieusement plus

courte que la largeur : cela ne dérange en aucune façon notre ordinateur, car ne sachant pas ce qu'est une longueur ni une largeur, il serait bien incapable d'y trouver une erreur. Ne vous attendez pas ici à un SYNTAX ERROR. Ce n'est pas une erreur de programmation mais une erreur quant au contenu d'une valeur, donc a priori indécelable par le système.

Ex. n°4 :

Il ne se passe rien de rien. Vraiment ? Et bien si ! Sauf une petite chose : le ok s'affiche maintenant TROIS lignes plus bas.

Ex. n°5 :

Deux lignes, deux erreurs !

— SYNTAX ERROR en 20 ou PRINT et TVA sont collés
— idem ligne suivante :

```
10 PRINT TVA
20 PRINT PTTC
```

Ceci est syntaxiquement correct. L'inconvénient, c'est que TVA et PTTC n'ont pas de valeur, donc l'ordinateur va afficher deux fois 0.

2.
LA RIGUEUR ET L'ANALYSE

Nous venons de voir quelques manipulations de l'ordinateur et appris à faire de petits programmes. On comprend bien qu'en réalité, la programmation est un peu plus difficile que cela. Pour bien analyser et résoudre un problème complexe il faut suivre une certaine rigueur ; or, cette rigueur est bien souvent absente et beaucoup de programmes construits par des débutants pêchent souvent par leur longueur inutile, la redondance de certaines parties, leur inélégance ou bien plus prosaïquement leurs erreurs logiques.

Il faut donc acquérir cette rigueur nécessaire. Comment ?

Il existe une méthode qui se propose de décomposer le problème à analyser en trois parties :

L'algorithme
L'organigramme
L'écriture du programme.

En réalité, il ne sera pas nécessaire de toujours utiliser cette méthode. Cependant, les débutants auront intérêt à la faire : l'expérience venant, ils apprendront à se passer de la première ou de la seconde partie pour écrire directement le programme ; encore que ceci n'est pas toujours vrai dans le cas de problèmes très complexes. Nous conseillerons pour notre part de lire attentivement ce qui suit.

2.1. L'ALGORITHME

Ce n'est pas à proprement parler une méthode d'analyse appartenant à l'informatique. Ce terme, qui est une déformation du nom du mathématicien arabe Al-Kharezmi (IX^{e} siècle), signifie que la solution d'un problème se situe dans le raisonnement qui décrit les étapes nécessaires à la résolution de ce problème. Prenons un exemple apparemment trivial.

Soit le problème suivant :

Vous désirez laver votre linge : ceci est votre but. Pour parvenir à la solution, soit obtenir du linge propre, vous pratiquez un algorithme de ce genre :

Premièrement : je prends mon linge sale,
deuxièmement : j'entre ce linge dans la machine à laver,
troisièmement : j'entre de la lessive dans la machine à laver,
quatrièmement : je choisis le bon programme (!),
cinquièmement : je démarre la machine à laver,
sixièmement : je reviens un certain temps plus tard : j'ai du linge propre ; le problème est résolu.

Bien souvent, les personnes lisant ou écoutant ce type d'analyse ironisent : jamais on ne se dit cela quand on veut laver du linge ! C'est en partie vrai mais en réalité cet algorithme précis est intériorisé de telle manière qu'on n'a plus conscience de le suivre et pourtant, il a bien fallu l'apprendre à un moment donné : personne ne vient au monde en sachant utiliser une machine à laver ! Et ceci est vrai pour beaucoup d'actes banals et quotidiens. Bref, nous sommes comme monsieur Jourdain, le bourgeois gentilhomme : nous faisons des algorithmes sans le savoir !

Revoyons de près cet algorithme. On s'aperçoit qu'il serait possible de le détailler encore plus : c'est d'ailleurs à conseiller.

Plus intéressant, on s'aperçoit que certaines actions sont permutables, d'autres non. Ainsi, en est-il de la deuxième et troisième partie : on peut parfaitement introduire d'abord la lessive puis le linge. Par contre, il est impossible de démarrer d'abord le programme et d'entrer ensuite le linge. On s'aperçoit donc qu'un ordre rigoureux en général s'impose si on veut aboutir à la résolution de notre problème. C'est là toute la science de l'algorithme : seule une analyse rigoureuse et rationnelle permet de trouver la solution. Ceci implique une évidence : il faut connaître, avant de programmer, la méthode à appliquer, ce qui ramène d'ailleurs l'ordinateur à sa vraie place. Il n'est

pas capable de trouver pour vous comment résoudre un problème : c'est vous-même qui devez le trouver. Il est donc parfaitement inutile de demander à l'ordinateur votre âge ou le nom de votre grand-père. Il ne répondra que si vous l'avez programmé en ce sens.

Regardons encore de plus près cet algorithme. On voit que chaque phase (ou phrase) est numérotée, comme dans les programmes précédents : premièrement, deuxièmement, etc. On voit aussi que chacune de ces phrases comporte un verbe, entre prends choisis démarre, qui détermine une action précise et qui sont suivis d'un ou plusieurs arguments, linge sale, programme lessive, linge propre. C'est exactement ce que nous avons fait dans les exercices précédents :

— ENTRE est devenu en Basic INPUT,
— DEMARRE est devenu RUN, par exemple,
— Les arguments y étaient PHT, TVA ou PTTC,
— Les actions nécessaires étaient numérotées.

Ainsi voit-on la parenté qu'il y a, d'un point de vue purement formel, entre calculer une TVA et laver son linge ! Plus sérieusement, on a bien compris qu'un problème se résoud relativement simplement si l'on décompose ce problème en petits morceaux plus faciles à comprendre, donc à analyser, donc à formaliser en Basic.

Tout ceci est donc globalement appelé "algorithme". Nous savons cependant qu'il n'est pas si facile d'analyser un problème ; c'est pourquoi nous allons voir maintenant une méthode permettant de structurer l'analyse et de faciliter la création de l'algorithme.

Il faut noter que cette méthode n'est pas toujours aisée à appliquer et dépend de la complexité du problème. Cependant, elle va parfaitement être capable de vous aider dans la résolution de nombreux problèmes et en tout cas, de vous donner une base solide. A titre informatif, elle est largement inspirée sur la structuration du langage PASCAL.

Reprenons un instant l'exemple du calcul du Prix TTC. Dans ce problème nous avions défini trois valeurs :

La TVA, égale à 18,6, le prix hors-taxes, PHT et le prix TTC, appelé PTTC et égal à : PHT + (PHT * TVA/100).

Ces trois valeurs ont des ressemblances et des différences. Elles se ressemblent car elles sont des valeurs numériques, c'est-à-dire qu'elles symbolisent des nombres. Elles sont différentes en ce sens que : la première, TVA, a une valeur connue et constante, 18,6 ; en revanche, la seconde n'a pas, a priori, de valeur précise, mais est susceptible de prendre toutes les valeurs possibles, donc de varier.

La troisième, PTTC, réunit les caractéristiques des précédentes : elle est à la fois connue, car on sait la calculer selon une formule, mais reste quand même inconnue, car dépendante de la connaissance des deux précédentes. En d'autres termes, on vient de définir ici les deux types de valeurs possibles : les valeurs constantes et les valeurs variables, plus souvent appelées constantes et variables. De plus, une variable peut être simple, comme PHT, ou issue d'un calcul, comme PTTC.

Ainsi, lors de la formation de l'algorithme, nous nous emploierons à chercher les constantes et les variables. Nous avons également et simultanément accompli autre chose : nous avons NOMMÉ les valeurs en question, TVA pour la tva, PHT pour le prix hors-taxes et PTTC pour le prix toutes taxes comprises.

Enfin, nous avons été amenés à trouver une formule de calcul :

PTTC = PHT + (PHT * TVA/100)

Donc, premier principe :

Recherche et appellation des constantes, des variables et des formules permettant les calculs puis affichage des résultats. Par exemple, pour faire l'algorithme permettant de calculer la surface d'un cercle, nous pourrions procéder ainsi :

Recherche et appellation des constantes, variables et formules :

On se souvient de la formule de calcul : surface = PI * rayon * rayon

Recherche des constantes :

La valeur PI est une constante égale à 3.14 environ ;

Recherche des variables :

La valeur rayon ;

La valeur surface :

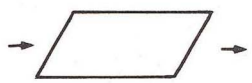
Surface = PI * rayon * rayon

(Nous conseillerons de faire l'algorithme en écrivant sur un brouillon toutes les idées qui viennent au fur et à mesure que l'on découvre quelque chose de nouveau. L'algorithme, comme d'ailleurs l'organigramme, n'est qu'une étape intermédiaire, pas une œuvre d'art !).

Ainsi, pour faire l'algorithme, il a suffit de chercher à décomposer le problème, de se dire simplement comment on trouve la solution et avec quelles valeurs. De plus, on peut numéroter les étapes.

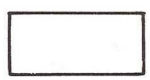
2.2. L'ORGANIGRAMME

Là aussi, nous n'employons pas une notion nouvelle. L'organigramme n'est que la traduction imagée et simplifiée de l'algorithme. Nous allons symboliser chaque étape du traitement de l'information par des figures adaptées. Ainsi, l'entrée ou la sortie d'une information sera figurée par un parallélogramme :

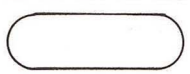


Une petite flèche pourra préciser l'entrée ou la sortie, selon qu'elle sera écrite à gauche ou à droite de la figure.

Le traitement effectif, un calcul par exemple, sera figuré par un rectangle



Le début et la fin le seront par :

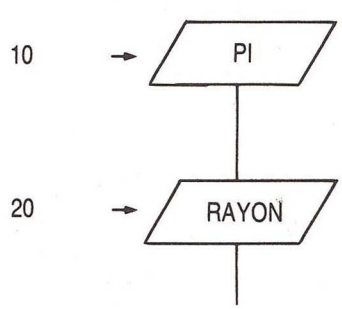


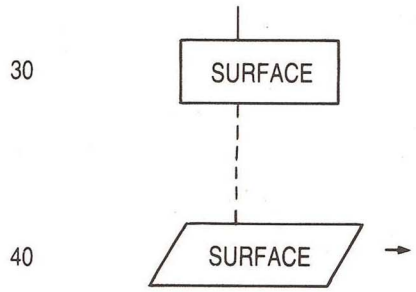
L'intérieur de la figure pourra contenir une information brève quant au traitement symbolisé. Evidemment, il n'est pas nécessaire d'y entrer toutes les informations déjà écrites lors de l'algorithme : cela ferait double emploi inutilement.

Pour symboliser le transfert de l'information, nous relierons les figures par des flèches ou une simple ligne.

Nous pourrons de plus numéroter les figures de 10 en 10, par exemple...

Pour reprendre l'algorithme précédent, nous ferions l'organigramme de la manière suivante :





2.3. LA PROGRAMMATION

Arrivés à ce stade, il ne nous reste plus qu'à traduire l'algorithme et l'organigramme dans le langage choisi, pour nous le Basic.

```
10 PI = 3.14
20 INPUT "entrez le rayon SVP " ;  rayon
30 SURFACE = PI * rayon * rayon
40 PRINT "la surface du cercle est de " ; SURFACE
```

Résumons donc les trois étapes de notre calcul de surface :

1 — Nous faisons l'algorithme en cherchant, nommant et en formulant les différentes constantes et variables : nous les écrivons dans l'ordre de traitement supposé juste (souvenez-vous : on met d'abord le linge dans la machine et on démarre ensuite la machine, pas le contraire !). Pour calculer la surface, il faut d'abord connaître PI, et connaître la formule.

2 — Nous transformons l'algorithme en une série de figures symbolisant les étapes de l'algorithme et symbolisant en même temps les lignes du futur programme.

3 — Nous traduisons en Basic les informations précédentes.

Exercices sur ce chapitre

Voici différentes formules permettant le calcul de surface ou de volume de plusieurs figures. Faites le programme correspondant, sans oublier l'algorithme et l'organigramme.

Surface du trapèze :

surface = petit côté + grand côté / 2 * hauteur

Surface du triangle équilatéral :

surface = côté ^{2} / 4 * $\sqrt{3}$

Pour extraire la racine carrée d'un nombre, par exemple 3, on fait :

PRINT SQR(3) : de square root, racine carrée.

Pour élever à une puissance, on utilise la flèche verticale, symbolysée par l'accent circonflexe ^ :

PRINT 2 ^ donne 8.

Surface du pentagone :

surface = 5 / 8 * rayon ^{2} * $\sqrt{10 + 2 * \sqrt{5}}$

Surface de l'ellipse :

surface = PI / 4 * grand diamètre * petit diamètre

Volume d'un cube :

volume = côté ^{3}

Volume d'une sphère

volume = 4 / 3 * PI * rayon ^{3}

Solutions commentées

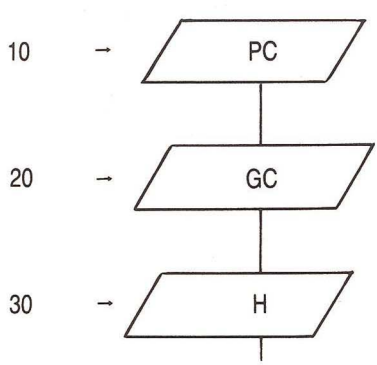
Surface du trapèze :

Algorithme :

Constantes : aucune

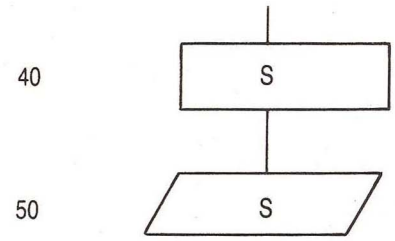
Variables : PC, pour petit côté
GC, pour grand côté
H, pour hauteur
S pour surface

Organigramme :



Programme

```
10 INPUT "entrez le PC " ; PC
20 INPUT "entrez le GC " ; GC
30 INPUT "entrez la hauteur" ; H
```




```
40 S = (PC + GC) / 2 * H
50  PRINT "la surface est " ;S
```

Surface du triangle équilatéral :

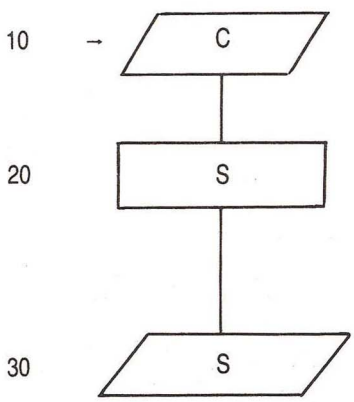
Algorithme

Constantes : aucune

Variables : C pour côté
S pour surface

Organigramme

Programme




```
10  INPUT "entrez le côté " ;  C
20  S = (C ^ 2) / 4 * SQR (3)
30  PRINT "la surface est " ; S
```

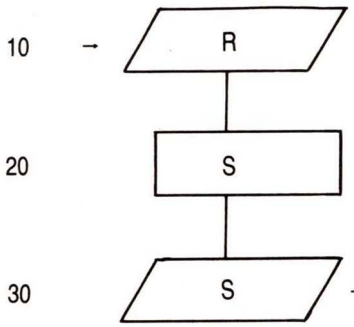
Surface du pentagone :

Algorithme

Constantes : aucune

variables : R pour rayon
S pour surface

Organigramme




```
10  INPUT "entrez le rayon " ; R
20  S = 5 / 8 * R ^2 * SQR (10 +
                2 * SQR (5))
30  PRINT " la surface est " ; S
```

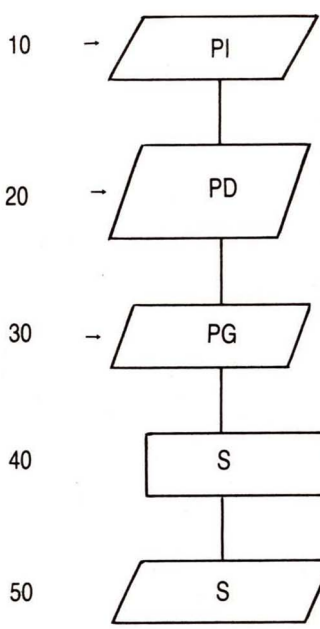
Surface de l'ellipse :

Algorithme

constantes : PI = 3.14

Variables : GD pour grand diamètre
PD pour petit diamètre
S pour surface

Organigramme



Programme

```
10 PI = 3.14
20 INPUT "entrez le petit diam " ; PD
30 INPUT "entrez le grand diam " ; GD
40 S = PI / 4 * PD * GD
50 PRINT " la surface est de " ; S
```

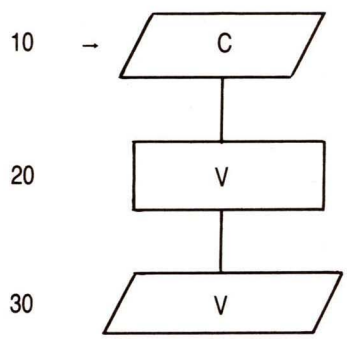
Volume du cube :

Algorithme

constantes : aucune

variables : C pour côté
V pour volume

Organigramme



Programme

```
10  INPUT "entrez le côté " ;  C
20  V = C ↑ 3
30  PRINT "le volume est de " ;  V
```

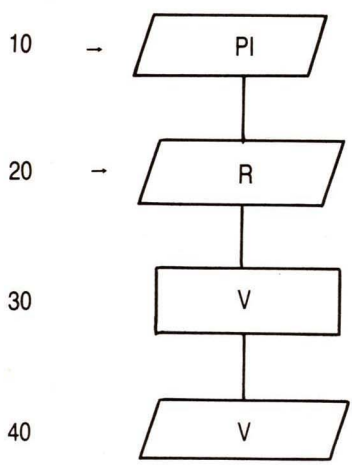
Volume de la sphère

Algorithme

Constantes : PI = 3.14

Variables :R pour rayon
V pour volume

Organigramme



Programme

```
10 PI = 3.14
20 INPUT "entrez le rayon " ;  R
30 V = 4 / 3 * PI *  R ↑ 3
40  PRINT "le volume est " ;  V
```

3.
FONCTIONS ET INSTRUCTIONS ELEMENTAIRES

3.1. LA DISQUETTE

3.1.1. Formatage

Une fonction particulièrement intéressante, après avoir fait tous les programmes précédents, consiste à les sauvegarder sur disque, ceci dans le but de pouvoir les utiliser ultérieurement.

Cependant, il faut s'assurer que votre disquette est prête pour cette opération. Si elle n'a jamais servie, elle est vraisemblablement non-formatée. Le formatage est une opération obligatoire et simple à effectuer : il faut pour cela être dans le système d'exploitation, le CP/M, dont nous parlions au début de cet ouvrage. Si vous êtes sous BASIC, il faut d'abord faire SYSTEM : attention, ce faisant, vous quittez le basic, et le programme que vous aviez disparait. Une fois le signe A> affiché, vous devez avoir dans le lecteur la disquette où se trouve l'utilitaire DISCKIT ; pour le savoir faites DIR : le contenu de la disquette s'affiche sur l'écran ; pour formater la disquette, entrez DISCKIT puis suivez les instructions du CP/M. N'oubliez surtout pas que formater une disquette déjà formatée efface tous les programmes existant sur cette disquette.

3.1.2. Sauvegarde et utilisation des programmes

SAVE

Dès que votre disquette est prête à l'emploi, vous pouvez sauvegarder un programme avec la commande SAVE suivie du nom du programme ; ainsi, les programmes précédents peuvent être sauvés de la manière suivante :

SAVE ''TRAPEZE''

ou bien SAVE ''trapeze

Le dernier guillemet n'est pas obligatoire, et le Basic transforme les minuscules en majuscules.

Ainsi, vous donnez à votre programme le nom que vous voulez, du moment qu'il n'excède pas 8 caractères.

LOAD

Pour récupérer le programme ainsi sauvegardé, il suffit, après avoir introduit la disquette sur laquelle il est censé se trouver, d'entrer :

LOAD ''TRAPEZE'' ou

LOAD ''trapeze.

Il suffit, si le message ok apparaît, de faire LIST et de voir de nouveau le programme prêt à l'emploi.

La demande d'un programme non-existant provoque le message d'erreur ''FILE NOT FOUND'', ''FICHIER NON TROUVE''.

FILES, DIR

Pour vérifier si le programme se trouve sur la disquette, on peut faire FILES ou DIR : ceci affiche la liste complète des programmes enregistrés. L'écran affiche le nom de chaque programme suivi de l'extension ''.BAS''; ce terme signifie que le programme est en ...BASic et est ajouté automatiquement par le ...BASIC.

KILL

Si un programme n'a plus notre faveur, il est tout à fait possible de le tuer ! Avec l'ordre KILL (tuer en anglais), on se débarasse proprement et froidement du programme superflu :

KILL ''TRAPEZE.BAS

3.2. VALEURS ALPHANUMERIQUES

Pour l'instant, nous n'avons utilisé que des nombres. Ces nombres étaient représentés ou bien directement en tant que nombres ou bien en tant que valeurs constantes et variables. Il est cependant possible de stocker en mémoire des valeurs constantes et variables alphanumériques. Ce dernier terme signifie simplement tout ce qui n'est pas numérique. Ainsi, le mot ''DUPOND'' ou le mot ''DURAND'' ne peuvent être des nombres, ou des valeurs symbolisant des nombres, car écrits entre guillemets. Tout ensemble de caractères écrit de cette manière est appelé une chaîne de caractères, ou chaîne alphanumérique. Il faut bien comprendre cette notion : si 123 est un nombre composé des chiffres 1,2 et 3, ''123'' est une chaîne de caractères, composée des caractères ''1'', ''2'', et ''3''. Seuls les guillemets déterminent la différence entre les valeurs numériques et les valeurs alphanumériques. Comment indiquer précisément à l'ordinateur cela ?

On utilise le signe dollar, $, pour spécifier une valeur alphanumérique, comme dans l'exemple suivant :

A$ = ''DUPONT''

Ce faisant, on indique à l'ordinateur qu'une valeur, nommée A$, a pour contenu la chaîne de caractères DUPONT ; chaque fois que l'on fera, par exemple, PRINT A$, c'est bien entendu le mot DUPONT qui va s'afficher, sans les guillemets qui sont là uniquement pour spécifier ce caractère d'être alphanumérique.

On s'aperçoit que l'on peut utiliser les valeurs alphanumériques exactement comme les valeurs numériques. Bien sûr, il n'est pas possible de faire des opérations arithmétiques avec des noms, par exemple il serait bien difficile de diviser DUPONT par DURAND. En revanche, les notions de constantes ou de variables s'appliquent également aux valeurs alphanumériques. Supposons que, par exemple, nous fassions un très long programme, où il sera question 45 fois de l'expression :

''ENTREPRISE MACHIN, rue des Alouettes, 08200 SEDAN FRANCE''. Il est préférable d'introduire cette expression en constante dès le début du programme, par exemple comme ceci :

NOM$ = ''ENTREPRISE MACHIN, rue des Alouettes, 08200 SEDAN FRANCE'', puis quand c'est nécessaire, d'utiliser seulement la constante NOM$ plutôt que son contenu. Il est beaucoup plus facile d'utiliser une expression par son nom symbolique, plutôt que par un contenu fastidieux à écrire. De plus, si l'entreprise change d'adresse

et qu'il faut donc corriger le programme, il est nettement plus aisé de corriger UNE ligne, celle où se trouve la constante en question, que 45 autres lignes, dispersées dans un programme très long. Il va sans dire que cette remarque s'applique également aux constantes numériques.

Nous disions que les valeurs alphanumériques s'utilisaient aussi comme les valeurs numériques. On se souvient que pour entrer une variable numérique, on se servait de INPUT. Pour entrer dans PHT, par exemple, on faisait :

INPUT PHT, ou bien

INPUT ''entrez le prix hors-taxes SVP'' ; PHT

On peut donc faire de même avec NOM$; par exemple, si l'on désire entrer le nom de monsieur DUPONT en mémoire :

INPUT NOM$, ou bien

INPUT ''entrez le nom de la personne'' ; NOM$

La syntaxe est rigoureusement la même. Nous avons appris comment faire un algorithme ; il suffit de préciser, outre la recherche des constantes numériques, celle des constantes alphanumériques, et outre la recherche des variables numériques, celle des variables alphanumériques.

Testons tout ceci avec quelques exemples.

On voudrait faire le programme qui demande à l'utilisateur le nom, l'âge, et le numéro de téléphone d'une personne, puis qui affiche l'ensemble des données. Analysons le problème et faisons l'algorithme.

Recherche des constantes :

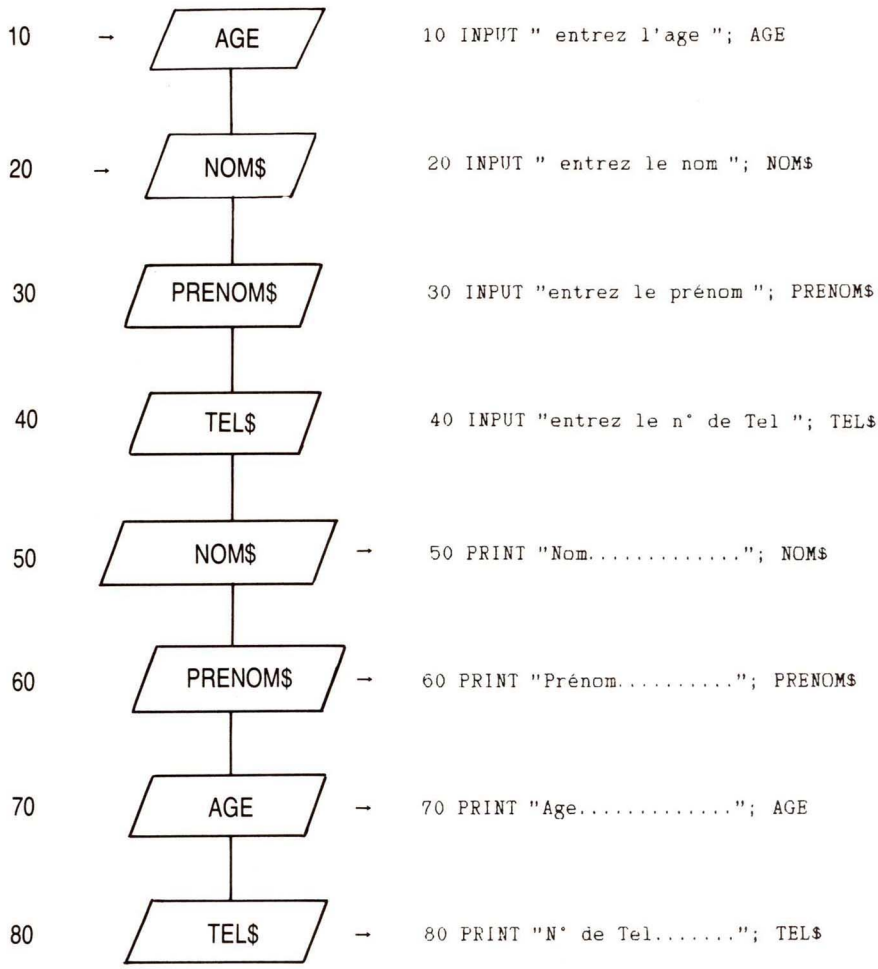
numériques : aucune
alphanumériques : aucune

Recherches des variables :

numériques : AGE, pour l'âge ;
alphanumériques : NOM$, pour le nom ;
PRENOM$, pour le prénom
TEL$, pour le téléphone.

Commentaires : le n° de téléphone n'est pas un nombre, car susceptible d'être écrit ainsi : 12.12.23.34, par exemple ; or, il est impossible d'écrire un nombre avec TROIS virgules, ici points décimaux.

L'organigramme et le programme ne posent pas de problème spécifique :



Commentaires :

Rien à dire, sauf peut-être la façon dont on peut afficher des résultats de manière présentable.

Autre exemple :

Proposons-nous la création d'un programme permettant une facture EDF très simplifiée ; il devra demander le nom et l'adresse du client, sa consommation en kilowatts/heures des heures pleines et creuses, puis afficher sur l'écran la facture correspondante, par exemple comme ceci, et sans oublier la TVA :

```
************************************
EDF-GDF
rue Ampère 75000 PARIS
client : Mr Dupont louis
         30, rue des Alouettes
         75010 PARIS
------------------------------------
Cout H/Pleines:0.50
Cout H/creuses:0.30

Heures pleines: 340
Heures creuses: 200

Cons H/P      : 170
Cons H/C      : 60

ToT.Hors Taxes: 230

TVA  18.6%    :42.78

Somme due :    272.78
************************************
```

Algorithme :

Recherches des constantes :

numériques : TVA = 18.6
CHP = 0.50 coût du KW/H heure pleine
CHC = 0.30 coût du KW/H heure creuse

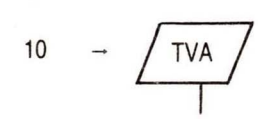
alphanumériques : EDF$ = ''EDF-GDF rue Ampère 75000 Paris''

Recherche des variables :

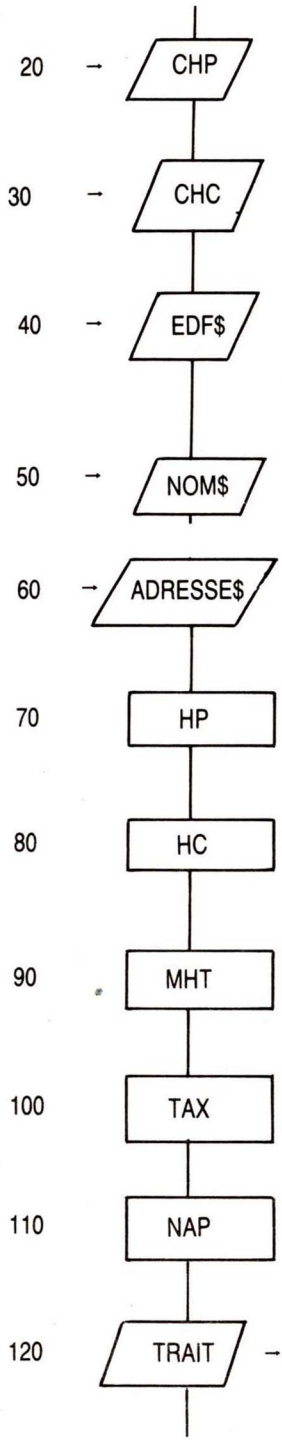
numériques : KHP : consommation en KW/H heures pleines du client
KHC : consommation en KW/H heures creuses du client
HP = CHP * KHP : total à payer heures pleines
HC = CHC * KHC : total à payer heures creuses
MHT = HP + HC : montant hors taxes
TAX = MHT + (MHT * TVA / 100) : montant de la TVA
NAP = MHT + TAX : net à payer.

alphanumériques : NOM$: nom du client
ADRESSE$: adresse du client.

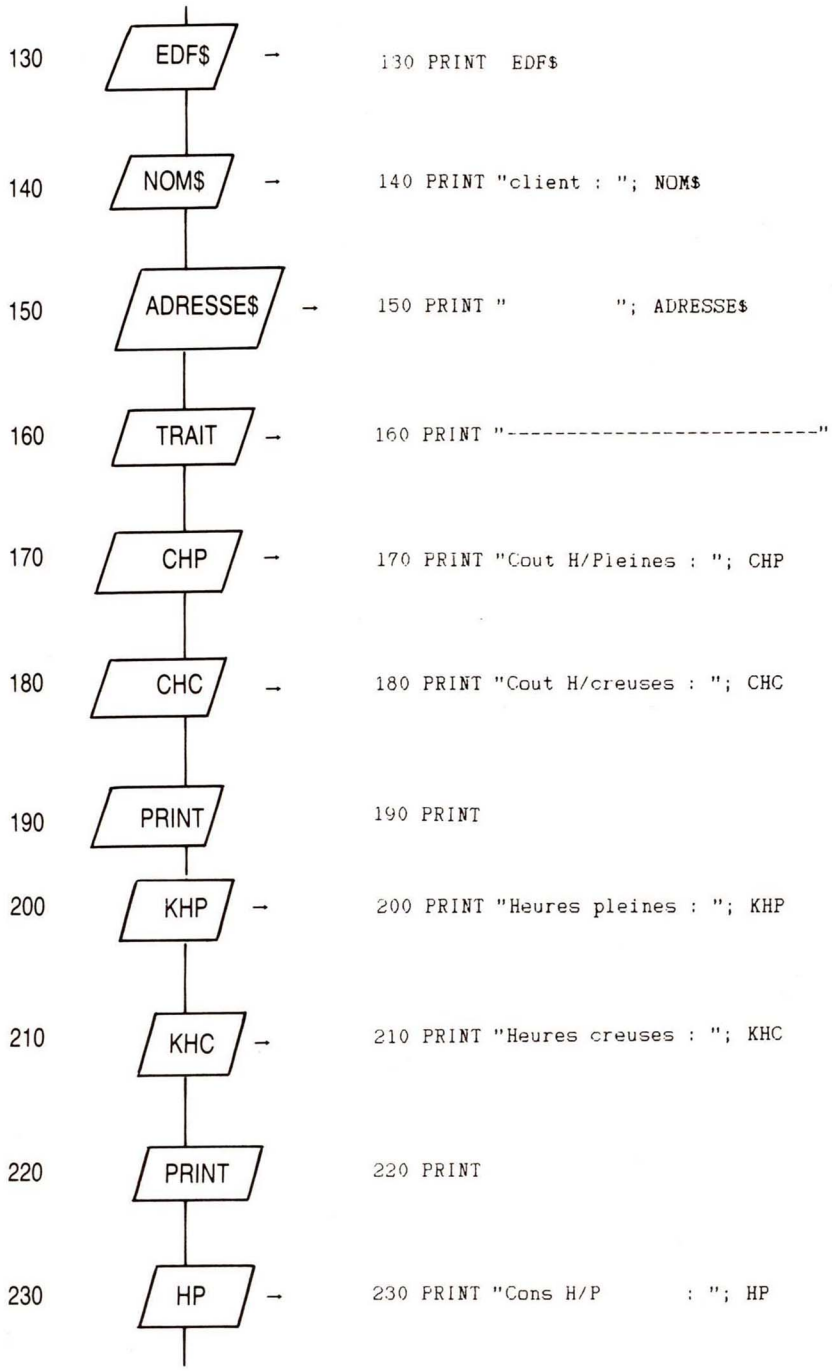
Organigramme

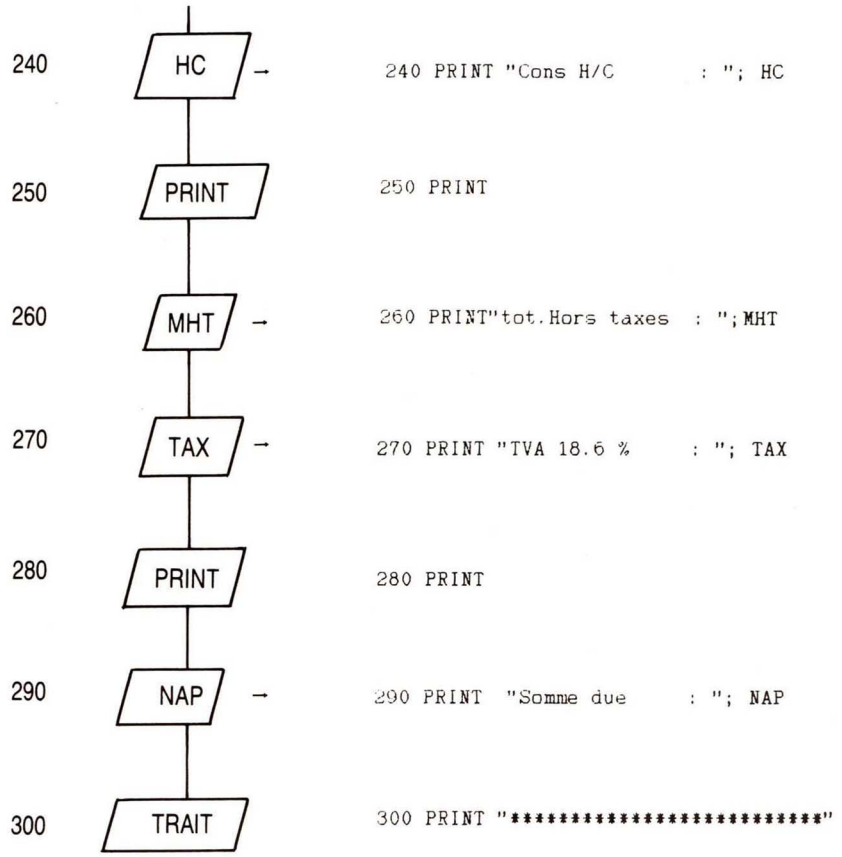



```
10 TVA = 18.6
```




```
20 CHP = 0.5

30 CHC = 0.3

40 EDF$= "EDF-GDF rue Ampère 75000
PARIS"

50 INPUT "Nom du client "; NOM$

60 INPUT "Adresse du client";ADRESSE$

70 HP = KHP * CHP

80 HC = KHC * CHC

90 MHT = HP + HC

100 TAX = MHT + ( MHT * TVA / 100 )

110 NAP = MHT + TAX

120 PRINT "*************************"
```


130
EDF$
130 PRINT EDF$
140
NOM$
140 PRINT "client : "; NOM$
150
ADRESSE$
150 PRINT " "; ADRESSE$
160
TRAIT
160 PRINT "--------------------------"
170
CHP
170 PRINT "Cout H/Pleines : "; CHP
180
CHC
180 PRINT "Cout H/creuses : "; CHC
190
PRINT
190 PRINT
200
KHP
200 PRINT "Heures pleines : "; KHP
210
KHC
210 PRINT "Heures creuses : "; KHC
220
PRINT
220 PRINT
230
HP
230 PRINT "Cons H/P : "; HP



3.3. UTILISATION DE L'IMPRIMANTE ET ECRITURE DU PROGRAMME

LPRINT, LLIST

Une facture s'édite rarement sur un écran, plus souvent sur papier. Pour écrire avec l'imprimante, il suffit de transformer tous les PRINT du programme en LPRINT. De plus, il peut s'avérer utile de lister les programmes sur papier : on emploiera alors LLIST.

Les deux points '':''

On peut parfaitement écrire plusieurs instructions sur la même ligne. Pour que le Basic comprenne cependant qu'il s'agit d'instructions différentes, on les sépare par deux points '':'' Ainsi :

10 PRINT ''BONJOUR'' : PRINT ''au revoir'' : PRINT ''c'est pas malin comme exemple '' : INPUT ''alors, entrez autre chose'' ; A$: PRINT A$

est identique à :

```
10 PRINT "bonjour"
20 PRINT "au revoir"
30 PRINT "c'est pas malin comme exemple"
40 INPUT "alors, entrez autre chose" ; A$
50 PRINT A$
```

3.4. AFFICHAGE

TAB

Bien souvent il est utile de pouvoir aligner sur l'écran ou sur l'imprimante des résultats ou des commentaires. Cela est possible avec l'instruction TAB, de tabulation. Elle s'utilise toujours avec PRINT suivie d'un argument indiquant la colonne choisie. Pour écrire sur la colonne 45, le nombre 45 on dira :

PRINT TAB(45) ; 45

Pour écrire A$ colonne 20, et sachant que A$ = ''DUPONT'', on dira :

PRINT TAB(20) ; A$

On aurait pu écrire aussi :

PRINT TAB(20) A$ ou bien encore

PRINT TAB(20) ''DUPONT''

Cette fonction sert à placer des valeurs où l'on veut sur l'écran mais du même coup sert également à aligner sur la gauche plusieurs valeurs successives. Ainsi :

```
10 PRINT TAB(20) ; 100
20 PRINT TAB(20) ; 1
30 PRINT TAB(20) ; 1000
```

donnera :

100
1
1000

Si les valeurs sont alphanumériques cela ne pose pas de problème ; en revanche, des valeurs numériques sont mal alignées sur la gauche ; il serait préférable de les aligner sur la droite. Cela est possible avec une autre fonction.

USING

C'est en conjonction avec TAB que l'on peut utiliser USING, pour obtenir l'alignement de nombres successifs sur leur droite, mais seulement de nombres.

Il faut au préalable définir le format d'affichage des nombres, selon le nombre de chiffres les composant. Ce format s'écrit avec le signe dièse, ''#''. Chaque dièse symbolise la place prise par chaque chiffre du nombre à afficher. Par exemple, si on désire aligner les nombres suivants : 123, 34,5, 10, 0.568, on va définir un format englobant au moins TROIS valeurs entières et TROIS valeurs décimales, parce que le plus grand nombre entier ici a trois chiffres et le plus grand nombre décimal (en taille) a également trois chiffres. Ainsi :

```
123
 34
  5
 10
  0.568
###.###
```

Pour pouvoir aligner ces nombres en colonne 35, par exemple, nous dirons :

```
10  PRINT TAB(35) USING "###.###";123
20  PRINT TAB(35) USING "###.###";34
30  PRINT TAB(35) USING "###.###";5
40  PRINT TAB(35) USING "###.###";10
50  PRINT TAB(35) USING "###.###";0.568
```

Cela donnera :

```
123.000
 34.000
  5.000
 10.000
  0.568
```

On notera que USING remplace les espaces vides à droite par des zéros.

La fonction USING permet en fait de réserver, à partir de la position établie par le TAB (ou bien à la première colonne sans plus de précision) un espace au nombre, mais un espace de type particulier où l'emplacement interne des chiffres est lui-même réservé par rapport au point décimal. On peut remarquer que les dièses étant des caractères, rien n'empêche de les considérer comme tels : on peut donc leur adjoindre d'autres caractères. Par exemple, le mot "francs" :

```
10 PRINT TAB(50) USING " ####.##  francs" ; 123
20 PRINT TAB(50) USING " ####.##  francs" ; 34.5
```

donnera :

```
123.00 francs
 34.50 francs
```

Insistons un peu sur la qualité de chaînes de caractères des dièses. Il est visible que, si l'on doit entrer toute une série de nombres, il faudra prévoir autant de formats avec USING. Il est plus simple de faire une valeur constante de tous ces dièses et les deux lignes précédentes pourraient se réécrire ainsi :

```
10 Z$ = "####.## francs "
20 PRINT TAB(50) USING Z$;123
30 PRINT TAB(50) USING Z$;34.5
```

De cette manière, non seulement le programme est plus court, donc plus rapide, moins gourmand en mémoire, diminuant les erreurs de frappe mais en plus permet la correction facile du format en cas d'erreur.

Une autre façon d'utiliser USING est de formater en une seule fois les nombres apparaissant sur la même ligne :

```
PRINT USING " * ####.##    *   ####.###   *   ###.#  *"; 123.45;345.4;12
PRINT USING " * ####.##    *   ####.###   *   ###.#  *"; 23; 34;23
```

donnera :

```
*  123.45    * 345.400   *  12.0 *
*   23.00    *  34.000   *  23.0 *
```

STRING$

Une possibilité existe pour écrire une chaîne de caractères identiques sans devoir les répéter, c'est la fonction STRING$, qui fort justement signifie chaîne (quel hasard !). Il suffit d'écrire le nombre de caractères et le caractère lui-même :

PRINT STRING$ (40, ''*'') donnera une chaîne de 40 astérisques :

```
****************************************
```

Comme c'est une chaîne, on peut en faire une constante alphanumérique ;

TRAIT$ = STRING$(40, ''*'')
et PRINT TRAIT$ donnera le même résultat que précédemment.

REM

Il est souhaitable bien souvent d'éclaircir un long listing d'explications ou simplement de mettre des titres à chaque partie importante du programme. On peut le faire avec l'instruction REM, de remark, qui n'est pas prise en compte par le Basic. Ainsi :

```
10 REM  bulletin de salaire
.........
.........
100 REM CALCUL SUR LES RETENUES
```

Les lignes 10 et 100 ici explicitent des parties mais le programme les ignore complètement : elles sont là exclusivement destinées au lecteur éventuel du listing mais ne servent à rien dans l'exécution du programme. Il est vivement conseillé d'en mettre, car il est particulièrement intéressant de pouvoir comprendre six mois plus tard un programme que l'on a complètement oublié : les lignes REM prennent alors toute leur valeur si elles sont abondantes et claires.

Une REM vide peut simplement servir à passer une ligne, pour aérer la présentation.

Exercice

Faire un bulletin de salaire ; il comprendra différents Taux de retenues :

Taux de maladie : 5,5 %
Taux de vieillesse : 4,7 %
Taux de veuvage : 1 %
Taux ASSEDIC : 1,72 %

Il portera sur un salarié dont on connaîtra le nom, l'adresse, le numéro de sécurité sociale, la qualification, le mois de salaire, le salaire et les primes, les acomptes, les indemnités éventuels. De plus, le bulletin de salaire sera agrémenté de lignes et de colonnes constituées d'astérisques.

Solution

Algorithme :

Recherches des constantes :

numériques :

les taux de retenues : MALADIE = 5.5
VIEILLESSE = 4.7
VEUVAGE = 1
ASSEDIC = 1.72

alphanumériques :

éventuellement les coordonnées de l'employeur. En fait, nous pouvons intégrer directement ces coordonnées dans la partie ''impression'' des résultats.

Y$ = ''# # # # # #.# # *''
Z$ = ''# # # # # #.# # *'' : format d'affichage des nombres
TRAIT$ = STRING$ (79, ''*'')
ETOILE$ = '' * *''
Traits divers d'affichage.

Recherches des variables :

numériques :

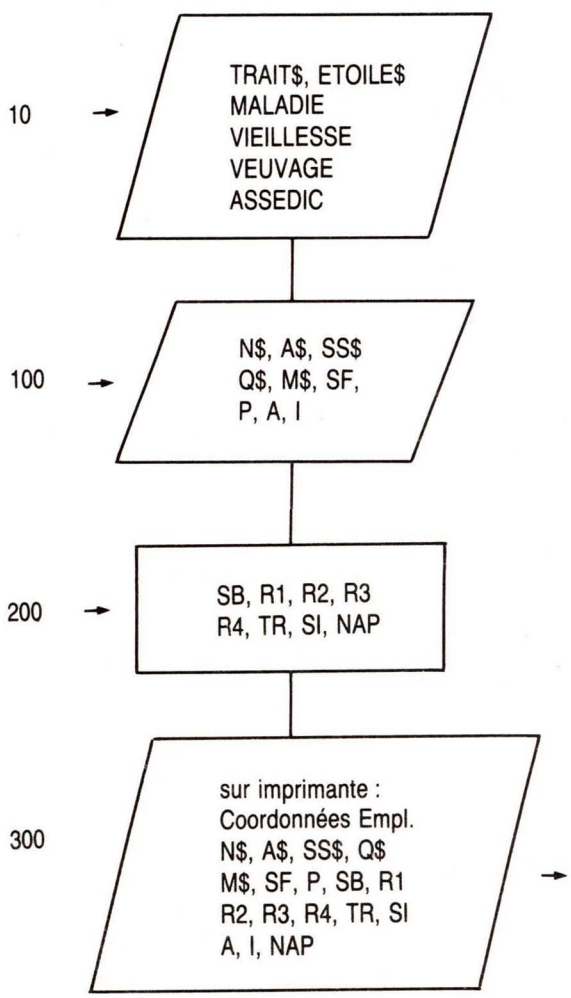
SF, salaire fixe
P, primes.
A, acomptes.
I, indemnités.
Calculs divers :
SB = SF + P : salaire brut.
R1, R2, R3, R4 : retenues.
R1 = SB * MALADIE / 100
R2 = SB * VIEILESSE / 100.
R3 = SB * VEUVAGE / 100.
R4 = SB * ASSEDIC / 100.
TR = R1 + R2 + R3 + R4 : total des retenues
SI = SB – TR : salaire imposable.
NAP = SI + I – A : net à payer.

alphanumériques :

N$: nom et prénom du salarié.
A$: adresse du salarié.
SS$: n° de sécurité sociale.
Q$: qualification du salarié.
M$: mois du salaire.

Organigramme

Pour aller plus vite tout en gardant l'esprit de la méthode, on pourra grouper les instructions de même nature en une seule figure :



Programme du bulletin de salaire et commentaires.

```
10 REM ---- BULLETIN DE SALAIRE --------------------
20 TRAIT$=STRING$(79,"*")
30 Y$="#####.##                                    *"
40 Z$="#####.##          *"
50 ETOILE$= "*
*"
60 MALADIE=5.5
70 VIEILLESSE=4.7
80 VEUVAGE=1
90 ASSEDIC=1.72
100 REM---------- SALARIE ----------------------
110 INPUT "entrez le nom du salarié.....";N$
120 INPUT "entrez son adresse.....";A$
130 INPUT "entrez son n° SS.....";SS$
140 INPUT "entrez sa qualification.....";Q$
150 INPUT "entrez le mois.....";M$
160 INPUT "entrez le salaire fixe.....";SF
170 INPUT "entrez les primes.....";P
180 INPUT "entrez les acomptes.....";A
190 INPUT "entrez les indemnités.....";I
200 REM------- CALCUL ----------------------------
210 SB=SF+P
220 R1=SB*MALADIE/100
230 R2=SB*VIEILLESSE/100
240 R3=SB*VEUVAGE/100
250 R4=SB*ASSEDIC/100
260 TR=R1+R2+R3+R4
270 SI=SB-TR
280 NAP=SI+I-A
290 REM
300 REM ------------IMPRESSION ------------------
310 LPRINT TRAIT$
320 LPRINT ETOILE$
330 LPRINT "* Société TRUC & MUCHE";TAB(79);"*"
340 LPRINT "* 1 rue des Alouettes 75002 PARIS";TAB(79);"*"
350 LPRINT ETOILE$
360 LPRINT TRAIT$
370 LPRINT "* mois: ";M$;TAB(79);"*"
380 LPRINT "* salarié.....";N$;TAB(79);"*"
390 LPRINT "* adresse.....";A$;TAB(79);"*"
400 LPRINT "* SS.....";SS$;TAB(79);"*"
410 LPRINT "* qualifiation..";Q$;TAB(79);"*"
420 LPRINT ETOILE$
```

```
430 LPRINT TRAIT$
440 LPRINT "* salaire fixe";TAB(60) USING Z$;SF
450 LPRINT "* primes";TAB(60) USING Z$;P
460 LPRINT "* salaire de base";TAB(60) USING Z$;SB
470 LPRINT ETOILE$
480 LPRINT "* retenues diverses";TAB(79)"*"
490 LPRINT "* maladie.....5.5% ";TAB(30) USING Y$;R1
500 LPRINT "* vieillesse..4.7% ";TAB(30) USING Y$;R2
510 LPRINT "* veuvage.....1.0% ";TAB(30) USING Y$;R3
520 LPRINT "* ASSEDIC.....1.72% ";TAB(30) USING Y$;R4
530 LPRINT ETOILE$
540 LPRINT "* total des retenues";TAB(30) USING Y$;TR
550 LPRINT ETOILE$
560 LPRINT "* salaire imposable";TAB(60) USING Z$;SI
570 LPRINT "* Acompte";TAB(60) USING Z$;A
580 LPRINT "* Indemnités";TAB(60) USING Z$;I
590 LPRINT ETOILE$
600 LPRINT "* NET A PAYER";TAB(60) USING Z$;NAP
610 LPRINT ETOILE$
620 LPRINT TRAIT$
```

Résultat :

```
********************************************************************************
* Société TRUC & MUCHE                                                         *
* 1 rue des Alouettes 75002 PARIS                                              *
*                                                                              *
********************************************************************************
* mois: Octobre                                                                *
* salarié..... DUPONT Charles                                                  *
* adresse..... 34 rue des Prunes 75005 PARIS                                   *
* SS.............. 1.34.08.08.456.123                                          *
*                                                                              *
********************************************************************************
* salaire fixe                                          7800.00                *
* primes                                                1000.00                *
* salaire de base                                       8800.00                *
*                                                                              *
* retenues diverses                                                            *
* maladie.....5.5%              484.00                                         *
* vieillesse..4.7%              413.60                                         *
* veuvage.....1.0%               88.00                                         *
* ASSEDIC.....1.72%             151.36                                         *
*                                                                              *
* total des retenues           1136.96                                         *
*                                                                              *
* salaire imposable                                     7663.04                *
* acompte                                                500.00                *
* indemnites                                             100.00                *
*                                                                              *
* NET A PAYER                                           7263.04                *
*                                                                              *
********************************************************************************
```

Commentaires :

On remarquera les deux constantes Y$ et Z$ qui servent à formater les nombres à l'affichage. Il y en a deux parce qu'il y a deux affichages possibles : les nombres sont ou à droite du bulletin ou au milieu ; ainsi, nous pouvons déterminer l'astérisque finale à droite.

Un petit truc pour transformer tous les PRINT en LPRINT et inversement. Au lieu d'utiliser EDIT, on se servira de AUTO.

4.
LES BRANCHEMENTS ET LES TESTS

4.1. BRANCHEMENTS INCONDITIONNELS

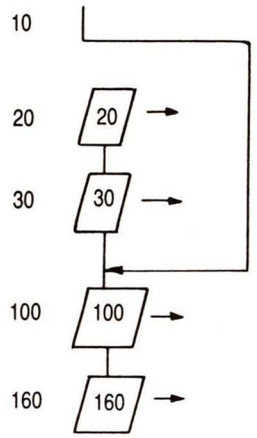
GOTO

Jusqu'à présent les programmes s'exécutaient de manière séquentielle, c'est-à-dire du début vers la fin. Ceci paraît d'ailleurs fort logique. Cependant, il peut s'avérer nécessaire de dérouter l'exécution normale du programme. Ainsi peut-on indiquer au système d'aller à une ligne spécifiée y continuer l'exécution avec l'ordre GOTO (de go to, aller à). Par exemple :

```
10 GOTO 100
20 PRINT 20
30 PRINT 30
100 PRINT 100
160 PRINT 160
```

Lors de l'exécution de ce programme, l'analyse par le Basic commence tout à fait normalement par la ligne 10 ; cette analyse rencontre l'instruction GOTO 100 qui lui intime l'ordre d'aller directement en ligne 100, sans passer par les lignes 20 et 30 ; on dit que le système *pointe* sur la ligne 100. Il exécute donc cette ligne et continue normalement la suite du programme. Et les lignes 20 et 30, direz-vous ?

Elles sont parfaitement et royalement ignorées par le système ! Nous allons pouvoir dessiner l'organigramme correspondant :

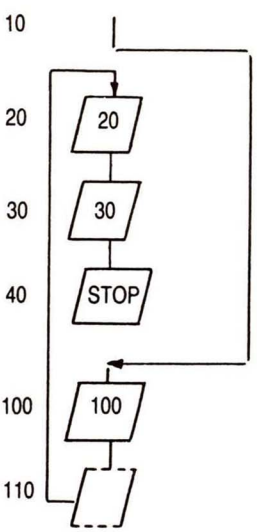


Quel en est l'intérêt ? A priori, aucun. En fait, si on sait que le pointeur peut remonter dans le programme, peut-être pourrait-on commencer à y trouver de l'intérêt, comme ceci, par exemple :

```
10 GOTO 100
20 PRINT 20
30 PRINT 30
40 STOP
100 PRINT 100
110 GOTO 20
```

L'instruction en 110 renvoie le pointeur en 20 et reprend donc l'exécution du programme, à la ligne suivant le premier GOTO.

On peut en dessiner l'organigramme de cette façon :



Vous n'êtes toujours pas convaincu de l'intérêt d'une telle fonction ? Attendez la suite...

4.2. BRANCHEMENTS CONDITIONNELS

IF... THEN

4.2.1. Première possibilité

Voici maintenant une des fonctions essentielles du Basic, celle qui permet de faire des choix, des tests, des prises de décision.

Il arrive souvent qu'une alternative ou qu'un choix multiple se présente dans un problème précis. Par exemple :

s'il pleut, je prends mon parapluie.

Reformulée, cette phrase pourrait être :

SI il pleut ALORS je prends mon parapluie.

L'alternative ici concerne le temps qu'il fait ; bien sûr en informatique les problèmes se posent de manière différente quant au fond.

SI... ALORS en Basic s'énonce ainsi : IF... THEN et s'utilise comme ceci :

Par exemple, un test peut être fait sur des valeurs données :

Nous avons A = 4 et B = 5 et nous disons :

SI A = B ALORS AFFICHER "A est pareil que B"

Peut-on afficher que A est pareil que B ? Non car visiblement A est différent de B : la condition requise, à savoir si A est égal à B, n'est pas remplie DONC il n'y a pas exécution de l'action prévue, à savoir afficher le message "A est pareil que B".

Traduisons quand même en Basic ces indications :

```
10  A = 4
20  B = 5
30  IF A = B THEN PRINT "A est pareil que B"
```

et faisons RUN :

Il ne se passe strictement rien de rien, ce qui n'est pas vraiment étonnant car A est toujours différent de B.

Changeons maintenant la ligne 10 en :

```
10 A = 5
```

et refaisons RUN :

Le message s'affiche : le test a réussi. On sait donc maintenant pratiquer des tests sur des valeurs numériques ou même alphanumériques si on sait utiliser les opérateurs nécessaires. Ainsi, il faut savoir que :

''inférieur à'' s'énonce avec le signe ''<''
''supérieur à'' s'énonce avec le signe ''>''
''supérieur ou égal'' s'énonce avec le signe ''= >''
''inférieur ou égal'' s'énonce avec le signe ''< =''
''différent de'' s'énonce avec le signe ''< >''

Nous pouvons donc réécrire le programme précédent en améliorant les comparaisons entre deux valeurs :

```
10  INPUT A
20  INPUT B
30  IF A = B THEN PRINT "A et B sont identiques"
40  IF A < B THEN PRINT "A est plus petit que B"
50  IF A > B THEN PRINT "A est plus grand que B"
```

Résumons cette instruction très puissante :

IF | CONDITION VRAIE | THEN | ACTION |

Si la condition est vraie, alors tout ce qui se trouve à droite de THEN est exécuté et le pointeur du programme passe *ensuite* à la ligne suivante.

Si la condition est fausse, alors le pointeur du programme passe *directement* à la ligne suivante.

Tout ce qui est entre IF et THEN est bien sûr du type relation logique (supérieur à, inférieur à, égal à, différent de). Par contre, tout ce qui est à droite de THEN peut être n'importe quelle instruction Basic, comme s'il y avait une ligne normalement constituée. Dans l'exemple précédent, on utilisait simplement l'instruction PRINT. On pourrait parfaitement y mettre des calculs à effectuer, des opérations diverses ou bien l'instruction GOTO, par exemple et ainsi faire des tests non pas seulement sur des valeurs propres au programme (comme dans l'exemple) mais également sur l'exécution du programme.

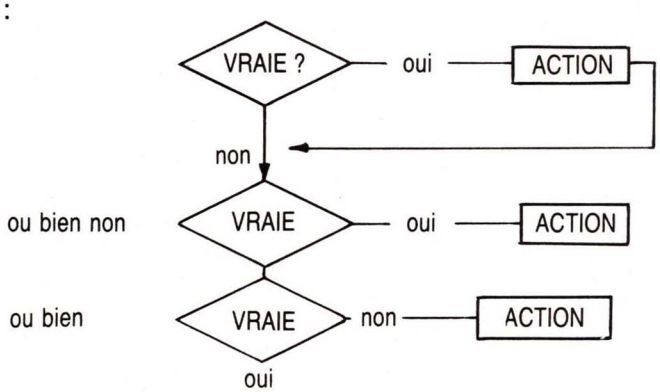
Ainsi, si l'on désire faire un programme permettant un choix dans son exécution :

```
10 PRINT " 1  ADDITION
20 PRINT " 2  SOUSTRACTION
30 PRINT " 3  MULTPLICATION
40 PRINT " 4  DIVISION
50 PRINT
60 INPUT "entrez votre choix "; C
70 IF C = 1 THEN GOTO 1000
80 IF C = 2 THEN GOTO 2000
90 IF C = 3 THEN GOTO 3000
100 IF C= 4 THEN GOTO 4000
200 REM
1000 REM ADDTION
1100 ..........
2000 REM SOUSTRACTION
2010 ..........
3000 REM MULTIPLICATION
3010 ..........
4000 REM DIVISION
4010 ..........
```

Ce programme (sans grand intérêt...) permet d'afficher un menu qui demande à l'utilisateur de choisir parmi quatre possibilités. Le choix détermine une bifurcation en lignes 1000, 2000, 3000 ou 4000. Là, une exécution quelconque (non figurée dans cet exemple) pourra être opérée. Ainsi, l'utilisateur du programme aura pu choisir parmi plusieurs alternatives.

Notons que l'expression ''IF... THEN GOTO 1000'' peut être simplifiée par ''IF.. THEN 1000''. Le GOTO est alors implicite et compris par le Basic.

Pour imager sur l'organigramme cette fonction, on utilise un losange :



Pour illustrer ces deux fonctions puissantes, proposons-nous la création du programme permettant de jouer au nombre mystérieux. Il existe à la radio le même jeu : une dizaine d'auditeurs propose à tour de rôle de découvrir une valeur inconnue d'eux. Chaque fois que le nombre proposé est plus grand ou plus petit que le nombre mystérieux le présentateur le spécifie et ainsi le joueur suivant bénéficie d'une fourchette sur laquelle il se fonde. Il va sans dire que pour les besoins de la démonstration, le nombre mystérieux est supposé être inconnu de la personne qui utilisera le programme !

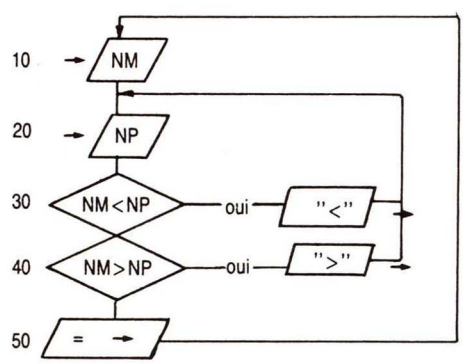
Algorithme :

Recherches de constantes numériques et alphanumériques : aucune

Recherches de variables numériques et alphénumériques : NM pour le nombre mystérieux et NP le nombre proposé.

Autre indication : Test vraisemblablement nécessaire.

Organigramme :




```
10 INPUT "nombre mystérieux";NM

20 INPUT "nombre proposé ";NP

30 IF NM < NP THEN PRINT "plus
   petit" : GOTO 20
40 IF NM > NP THEN PRINT "plus
   grand" : GOTO 20

50 PRINT "c'est gagné " : GOTO 10
```

Commentaires :

Tant que le nombre mystérieux n'est pas trouvé, le jeu doit indiquer si le joueur est au-dessus ou en-dessous de ce nombre, puis demander de nouveau au joueur de proposer une autre valeur (GOTO 20).

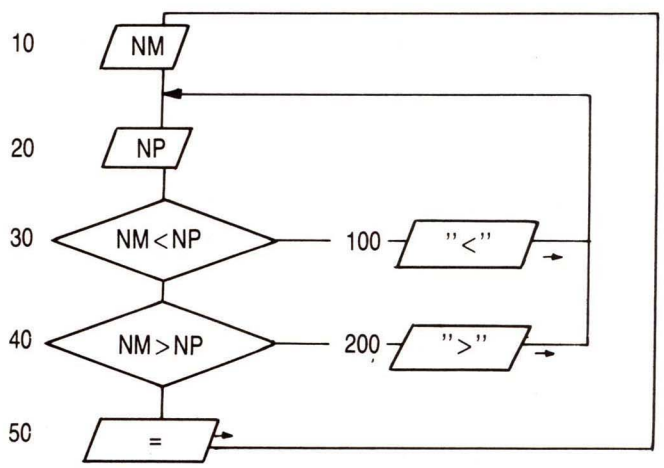
Il n'est pas nécessaire de tester l'égalité en ligne 50, car si le pointeur arrive sur cette ligne c'est qu'il a forcément échoué aux deux tests précédents. Le système repart ensuite en ligne 10 pour demander un autre nombre mystérieux.

Nous avons indiqué dans l'algorithme qu'un test était vraisemblable. En fait, c'est avec l'habitude que, face à de tels problèmes

nous finissons par acquérir de manière intuitive ce qui est vraisemblable de ce qui s'avère à l'usage intéressant. Cependant, chaque fois qu'un énoncé suggèrera un choix, il faudra maintenant penser au test, donc à IF et son compère THEN !

Variante possible :

Lors du dessin de l'organigramme, on peut se représenter autrement l'exemple précédent de la manière suivante, en notant après le dessin les numéros des figures.



L'organigramme est le même, seules les figures ont une numérotation différente, car elles ont été numérotées ainsi : on lit d'abord la colonne de figures de gauche, on les numérote. De même avec la colonne de droite, en prenant soin de choisir les numéros largement ; le programme sera donc différent :

```
10   INPUT "entrez le nombre mystérieux "; NM
20   INPUT "entrez le nombre proposé "; NP
30   IF NM < NP THEN 100
40   IF NM > NP THEN 200
50   PRINT " c'est gagné "
60   GOTO 10
100  PRINT "plus petit "
120  GOTO 20
200  PRINT "plus grand "
220  GOTO 20
```

On remarquera que cette dernière façon est relativement plus intéressante que la précédente car elle permet, le cas échéant, de faire

un très long développement des parties 100 et 200, alors que la première limitait à la longueur de la ligne les possibilités. Il faut savoir qu'une ligne ne peut excéder 256 caractères.

Une autre manière de représenter un choix multiple se fait simplement en imbriquant plusieurs IF... THEN dans la même ligne.

Ainsi :

IF condition-1 THEN IF condition-2 THEN IF condition-3 THEN action

L'action à suivre ne se fera que si les trois conditions sont vérifiées, elle ne se fera pas si une ou deux seulement sont vérifiées. Il faut cependant être très prudent dans l'usage de telles instructions :

IF NM > NP THEN IF NM < NP THEN PRINT ''les valeurs sont différentes. On aurait pu aussi bien utiliser < >, non ?''

4.2.2. Seconde possibilité

IF...THEN...ELSE

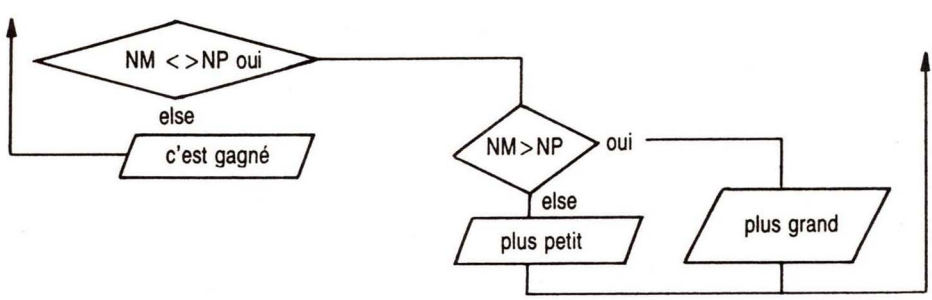
On peut encore représenter un test ou un choix avec une extension de l'instruction, ELSE (de else : autre, ici au sens de ''sinon'') :

SI CONDITION VRAIE ALORS ACTION-1 SINON ACTION-2

Ici, si la condition est vraie, l'action-1 est opérée puis le pointeur passe à la ligne suivante (sauf évidemment si l'action en question est un branchement à une ligne spécifiée).

Si la condition est fausse, non vérifiée, le pointeur passe à ou aux instructions spécifiées après ELSE. On peut aussi imbriquer des IF...THEN...ELSE, bien que cela ne soit pas très clair :

IF NM < > NP THEN IF NM > NP THEN PRINT ''plus grand'' : GOTO 20 ELSE PRINT ''plus petit'' : GOTO 20 ELSE PRINT ''c'est gagné'' : GOTO 10



Exercices sur les chapitres précédents

Ex. n° 1 :

Reprendre les programmes permettant de calculer des surfaces et en faire UN seul programme. Un menu général permettra de faire un choix parmi les calculs possibles, et sera de ce genre :

Calculs de surfaces et de volumes
1 rectangle
2 triangle
3 pentagone
4 ellipse
5 trapèze
Entrez votre choix SVP ?

Ex. n° 2 :

Que se passe-t-il lors de l'exécution des programmes suivants :

Programme n° 1

```
10 GOTO 100
20 PRINT 20
100 GOTO 20
120 PRINT 10
```

Programme n° 2

```
10  A = 5
20  B = 4
30  IF A > B THEN A = 3 : PRINT " A = 3" : ELSE  B = 9 :
    PRINT " B = 9 "
40  GOTO 30
```

Programme n° 3

```
10 A = 1
20 B = 4
30 C = 9
40 IF A = C THEN  B = C : C = A + B : PRINT A,B,C : GOTO 60
ELSE IF  B = C  THEN A = B :PRINT A,B,C ELSE PRINT A,B,C
50 A = C
60 GOTO 40
```

Ex. n° 3 :

RND INT

RND sert à créer un nombre aléatoire (de random, hasard) et s'utilise ainsi :

HASARD = RND fournit un nombre compris entre 0 et 1, ex 0.4356 ou bien avec un argument : RND(123).

INT (de integer, entier) sert à produire un nombre entier comme ceci :

PRINT INT (234.23) renvoie 234.

Comment utiliser ces deux fonctions pour créer un nombre aléatoire dans le jeu du nombre mystérieux ?

Solutions

```
10 REM ------------CALCULS DIVERS --------------------
20 REM
30 PRINT
40 PRINT " 1  RECTANGLE
50 PRINT " 2  TRIANGLE
60 PRINT " 3  PENTAGONE
70 PRINT " 4  ELLIPSE
80 PRINT " 5  TRAPEZE
90 PRINT
100 INPUT "entrez votre choix ";c
110 IF c=1 THEN 170
120 IF c=2 THEN 240
130 IF c=3 THEN 300
140 IF c=4 THEN 360
150 IF c=5 THEN 440
160 GOTO 10
170 REM
180 REM ------------ RECTANGLE ------------------------
190 REM
200 INPUT "entrez la longueur ";L
210 INPUT "entrez la largeur ";LA
220 PRINT "surface= ";L*LA
230 GOTO 10
240 REM
250 REM -----------TRIANGLE --------------------------
260 REM
270 INPUT "entrez le coté du triangle équilatéral";C
280 PRINT "surface="; ( C ↑ 2)/4 * SQR(3)
290 GOTO 10
300 REM
310 REM ----------PENTAGONE -------------------------
320 REM
330 INPUT "entrez le rayon";R
340 PRINT "surface=";5/8 * R↑2 * SQR(10 + 2* SQR(5))
350 GOTO 10
```

```
360 REM
370 REM ----------ELLIPSE-----------------------------
380 REM
390 PI=3.14
400 INPUT "entrez le petit diamètre ";PD
410 INPUT "entrez le grand diamètre";GP
420 PRINT "surface=";PI/4 * PD * GP
430 GOTO 10
440 REM
450 REM ----------TRAPEZE ----------------------------
460 REM
470 INPUT "entrez le petit coté";PC
480 INPUT "entrez le grand coté";GC
490 INPUT "entrez la hauteur ";H
500 PRINT "surface=";( PC+ GC)/2*H
510 GOTO 10
```

Commentaires : il n'y a rien à dire, sauf peut-être cette façon de découper le programme en plusieurs parties indépendantes. Nous en reparlerons plus précisément au chapitre 5.

Ex. n° 2 :

Programme n° 1 :

Le programme ne s'arrête plus et tourne seulement sur les lignes 20 et 10 ; la ligne 10 est utilisée une seule fois et la ligne 120 jamais.

Programme n° 2 :

L'écran affiche :

A = 3
B = 9 puis tourne indéfiniment en ligne 30 et 40.

Programme n° 3 :

Le programme tourne indéfiniment en affichant les valeurs suivantes :

1	4	9	
9	9	18	
9	9	18	
18	18	36	
18	18	36	etc.

Ex. n° 3 :

En reprenant la dernière version du jeu, on pouvait écrire :

10 NM = INT (RND * 1000)

Ce faisant, on crée un nombre entier aléatoire compris entre 0 et 1000.

5.
LOGIQUE, BOUCLES ET SOUS-PROGRAMMES

5.1. LES OPÉRATEURS LOGIQUES

OR

Nous venons de voir comment procéder à des tests. Nous pouvons améliorer encore cette possibilité avec les opérateurs logiques ; ainsi, nous pouvons dire lors de comparaisons, et en reprenant l'exemple du nombre mystérieux :

SI NM>NP OU <NP ALORS AFFICHE ''NM et NP sont différents''.

Traduit en Basic, cela donne :

IF NM>NP OR NM<NP THEN PRINT ''NP et NP sont différents''.

Vous avez remarqué que, si dans une phrase normale on ne répète pas sur quoi porte le test, il faut impérativement le faire en Basic : il serai faux d'écrire :

IF NM>NP OR <NP THEN etc.

De même que :

IF NM>NP OR IF NM<NP THEN etc.

Il est à noter que le test est vérifié si l'une ou l'autre des conditions est vérifiée. Il suffit que, OU NM>NP, OU NM<NP pour que le pointeur exécute ce qui suit THEN.

AND

On peut également utiliser le ET logique. Il faut remarquer qu'on appelle cet opérateur ainsi pour bien le distinguer du ET de liaison qu'on utilise souvent dans la langue française, quelquefois à tort. Par exemple, voici une liste de gens et leur adresse :

Jean Paris
Denise Sedan
Patrice Sedan
Pierre Charleville
François Sedan
Claude Paris

Question : quelles sont les personnes qui habitent Paris et Sedan ?

Réponse aucune. La bonne question eut été : quelles sont les personnes qui habitent Paris OU Sedan.

On écrira ainsi :

Par exemple :

IF A>B AND B>C THEN PRINT "A est plus grand que C"

A la différence du OU Logique, il faut que les *deux* conditions soient vérifiées pour exécuter la suite.

NOT

La négation est possible avec NOT, qui inverse l'état d'un argument.

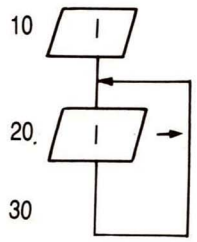
IF NOT A>B THEN PRINT "A plus petit que B".

5.2. L'ITÉRATION (1)

Nous allons voir maintenant une possibilité extrêmement intéressante du Basic, celle qui permet d'introduire un compteur dans un programme. Soit le programme suivant :

```
10 I = 0
20 PRINT I
30 GOTO 20
```

Si on lance le programme, il se met à tourner, à boucler comme on dit, sans jamais s'arrêter en affichant toujours la valeur de I, soit 0 :



Si nous voulons voir afficher des valeurs successives de I, par exemple 0, 1, 2, 3, etc., nous pouvons introduire la ligne suivante :

25 I = I + 1 et nous obtenons :

```
10 I = 0
20 PRINT I
25 I = I + 1
30 GOTO 20
```

Cette ligne est parfaitement bizarre d'un point de vue mathématique car on ne peut affirmer qu'une valeur est égale à elle-même plus 1 ! Si par exemple I vaut 5, nous obtenons 5 = 5 + 1, ce qui est manifestement louche. En réalité, nous savons maintenant pour l'avoir vu dans un chapitre précédent que le signe '' = '' en Basic, outre le fait qu'il signifie la même chose qu'en mathématique, symbolise une action précise :

Quand le système qui interprète la ligne rencontre une expression de ce genre, il évalue d'abord la partie à droite du signe égal. Or, que dit ici cette partie droite ? Elle dit simplement de prendre la valeur de I et de l'additionner à 1 ; comme I, en ligne 10, vaut 0, nous avons donc en ligne 25, l'équivalent de 0 + 1 qui a été évalué. Chacun sait que 0 + 1 doit donner 1, ce qui permet de dire que maintenant I = 1. Puis le pointeur passe en ligne 30 qui lui ordonne d'aller en ligne 20 ; là, le système affiche 1 puis de nouveau rencontre la ligne 25. De nouveau la partie droite de l'expression, I + 1, est évaluée ; or maintenant I vaut 1, nous venons de le voir, donc : 1 + 1 = 2, et 2 se met dans I :

I = 2

De nouveau encore le système va ainsi boucler indéfiniment jusqu'à ce qu'on l'arrête avec STOP et afficher les valeurs de I successives, de 1 en 1. Nous venons de faire un compteur. En effet, cette formulation, placée au bon endroit, va nous permettre de compter par exemple le nombre de fois où le pointeur repasse sur cette ligne et

d'en déduire éventuellement quelque chose. Reprenons l'exemple du jeu du nombre mystérieux et son homologue radiophonique. A la radio les auditeurs ne peuvent pas jouer indéfiniment ; le jeu se termine obligatoirement au bout de 10 joueurs. Tout se passe comme si le présentateur du jeu avait un compteur qui s'augmentait de 1 en 1 à chaque proposition de nombre, laquelle correspond à un joueur et que ce compteur était à chaque fois comparé au nombre 10 : tant que les nombres sont différents, le jeu continue, sinon il s'arrête et le présentateur dévoile au dernier candidat la valeur du nombre mystérieux... Comment intégrer ce compteur dans le programme ?

Algorithme : il faut que le compteur soit à zéro au départ du nombre mystérieux mais qu'il augmente (on dit qu'il s'incrémente) de 1 à chaque proposition. Ainsi, en reprenant cette dernière phrase, il suffit de trouver (et c'est facile) deux choses :

— Le départ du nombre mystérieux,
— chaque proposition.

Le départ est en ligne 10 : nous allons donc mettre COMPTEUR = 0 en ligne 15 ;

Chaque proposition est en ligne 20 : nous allons donc mettre COMPTEUR = COMPTEUR + 1 en ligne 25.

Mais il faut également qu'un test soit fait pour savoir si COMPTEUR est supérieur à 10, auquel cas le jeu s'arrête, le bon résultat est affiché et le jeu repart pour un autre nombre mystérieux (ici on dévie un peu du jeu radiophonique pour permettre de jouer avec l'ordinateur...). Le test doit donc être placé juste après l'incrémentation : c'est le meilleur endroit, ligne 27.

```
10  INPUT "entrez le nombre mystérieux "; NM
15  COMPTEUR = 0
20  INPUT "entrez le nombre proposé "; NP
25  COMPTEUR = COMPTEUR + 1
27  IF COMPTEUR > 10 THEN PRINT "c'est perdu : le bon
    résultat est ";NM: GOTO 10
30  IF NM < NP THEN 100
40  IF NM > NP THEN 200
50  PRINT " c'est gagné "
60  GOTO 10
100 PRINT "plus petit"
120 GOTO 20
200 PRINT "plus grand"
220 GOTO 20
```

Boucles :

Reprenons l'exemple décrivant une boucle infinie et généralisons le :

```
10  I = 0
20  PRINT I
30  I = I + 1
40  IF I < 10 THEN 20
50  SUITE DU PROGRAMME
```

Qu'avons-nous fait ici ? Simplement un compteur qui incrémente de 1 la valeur I, à partir de 0 (ligne 10) jusqu'à la valeur 10 (ligne 40). De cette manière, un programme quelconque nécessitant 10 répétitions (10 itérations) pourra être intégré entre le compteur et le test. Par exemple pour faire un calcul de factorielle. Si on désire calculer la factorielle de 5, on sait que :

5 ! = 1*2*3*4*5. On a bien ici les notions de compteur et de test car il faut incrémenter de 1 en 1 une variable 1 au départ, lui faire subir une multiplication et à chaque résultat intermédiaire vérifier si la variable en s'incrémentant reste toujours inférieure ou égale à 5, auquel cas le résultat final serait atteint. Ceci donne :

```
10 FACTORIELLE = 5
20 COMPTEUR = 1
30 RESULTAT = 1
40 RESULTAT = RESULTAT * COMPTEUR
50 COMPTEUR = COMPTEUR + 1
60 IF COMPTEUR <= FACTORIELLE THEN 40
70 PRINT "La factorielle de "; FACTORIELLE ; " est ";
          RESULTAT
```

Ici, nous créons en 40 la variable RESULTAT qui, à chaque passage du pointeur totalisera la multiplication du résultat précédent avec le nombre suivant de la liste ; comme au départ il n'y a pas de nombre précédent, on l'invente tout simplement en ligne 30. Le test en 60 permet de savoir s'il faut continuer ou afficher le résultat. Pour améliorer le programme, on remplacera la ligne 10 de la manière suivante :

10 INPUT "entrez la valeur";FACTORIELLE, ce qui permettra de calculer sur n'importe quelle valeur.

5.3. L'ITÉRATION (2)

5.3.1. Principes de la boucle

FOR...TO...NEXT

La possibilité de répéter autant de fois que l'on veut une action quelconque est très importante en programmation. C'est pour cela que les concepteurs du Basic ont prévu une instruction qui permet de faire exactement la même chose que ce que l'on vient de voir, et que l'on appelle une boucle.

Dans l'exemple du calcul de factorielle, on veut faire s'incrémenter de 1 en 1 une valeur nommée COMPTEUR, qui a pour valeur initiale 1 et pour valeur finale FACTORIELLE ; autrement dit, on fait varier une valeur COMPTEUR dans l'intervalle 1 à FACTORIELLE. On peut réécrire cette évolution ainsi :

```
10 INPUT "entrez la valeur ";FACTORIELLE
20 RESULTAT = 1
30 FOR COMPTEUR = 1  TO   FACTORIELLE
40 RESULTAT = RESULTAT * COMPTEUR
50 NEXT COMPTEUR
60 PRINT "la factorielle de ";FACTORIELLE ;" est ";
          RESULTAT
```

Ici, nous avons placé en ligne 30 l'intervalle de variation de la variable COMPTEUR avec les mots Basic :

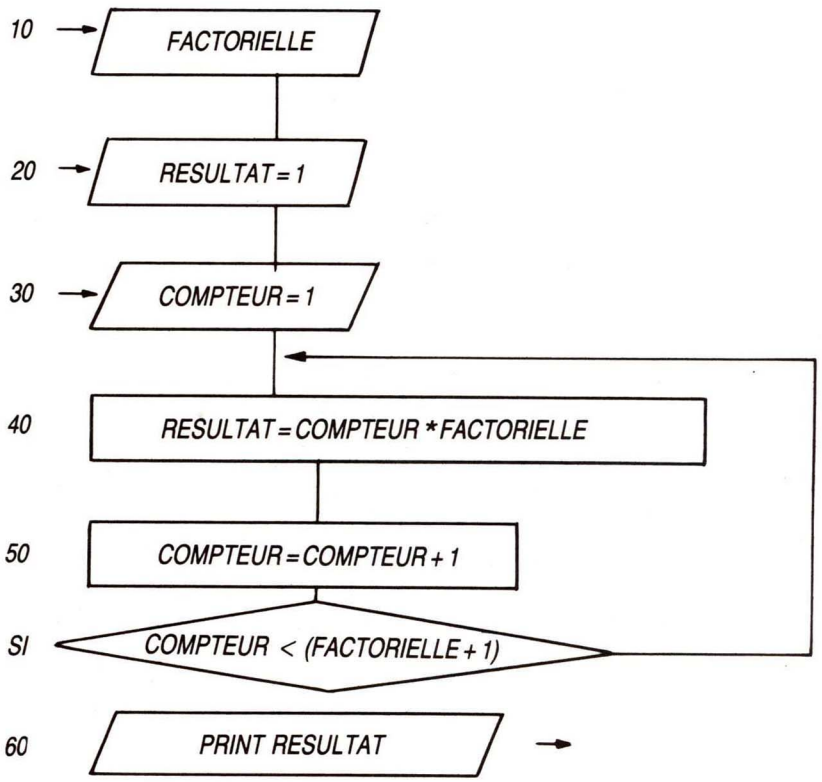
30 FOR COMPTEUR = 1 TO FACTORIELLE

30 POUR COMPTEUR = 1 JUSQU'A CE QUE COMPTEUR VALE FACTORIELLE. Il apparaît maintenant qu'avec cette nouvelle formulation il n'est plus nécessaire de préciser la valeur de l'incrémentation (qui était de 1 en 50) car la fonction FOR...TO le fait implicitement, automatiquement. De plus, cette manière de répéter une action quelconque est délimitée dans le programme par l'instruction de la ligne 50 :

50 NEXT COMPTEUR, qui signifie :

50 RETOURNER AU FOR précédent en augmentant de 1 la variable COMPTEUR et ceci tant que cette dernière n'est pas égale à la valeur spécifiée après le TO, donc ici FACTORIELLE. Puis, dès que la valeur spécifiée est atteinte, ne plus remonter dans la boucle mais passer à la suite du programme.

— Organigramme de FOR...TO...NEXT



Nous pouvons appliquer cette importante fonction de la manière suivante :

Proposons-nous de réécrire la table de multiplication de 5. Il suffit de programmer :

```
10 FOR I = 1 TO 10
20  PRINT  I ; " * 5 = " ; I * 5
30 NEXT I
```

Ici la boucle permet de simplifier l'écriture du programme. On aurait tout aussi bien pu écrire :

```
10 PRINT "1 * 5 = "; 1 * 5
20 PRINT "2 * 5 = "; 2 * 5
30 PRINT "3 * 5 = "; 3 * 5
40 PRINT "4 * 5 = "; 4 * 5
50 PRINT "5 * 5 = "; 5 * 5 etc...La différence saute aux
          yeux !
```

5.3.2. Boucles multiples

Outre la simplicité et l'élégance de l'écriture, la fonction FOR...TO...NEXT offre l'avantage de pouvoir s'imbriquer dans une fonction identique. Ainsi, si l'on désire TOUTES les tables de multiplication de 1 à 10, on peut écrire :

```
10  FOR J = 1 TO 10
20     FOR  I = 1 TO 10
30      PRINT J ; " * "; I ; " = " ; J * I
40     NEXT I
50  NEXT J
```

Dans cet exemple, la boucle J, qui se répète 10 fois, voit se répéter 10 fois la boucle I : il y a donc 10*10 = 100 calculs opérés. Vous noterez la manière particulière d'écrire le programme en décalant les lignes : ceci se nomme une indentation et n'a aucune incidence sur le déroulement du programme et sert simplement à mieux présenter un listing pour faire ressortir des aspects importants du programme.

On peut noter qu'il n'est pas obligatoire d'écrire le nom de la valeur après NEXT, même dans le cas de boucles multiples :

```
    ..........
    40  NEXT
50 NEXT
```

Ou bien on peut écrire deux boucles ainsi : NEXT I,J

IMPORTANT

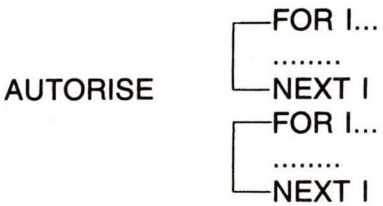
Dans le cas de boucles multiples, il ne faut pas qu'elles se croisent :

AUTORISE

```
FOR J...
  FOR I...
  .....
  NEXT I
NEXT J
```

INTERDIT

```
FOR J...
  FOR I...
  .....
  NEXT J...
NEXT I
```



5.3.3. Boucles avec paramètres

FOR...TO...STEP

On peut encore (!) améliorer la notion de boucle en l'étendant du mot STEP (de step, pas, écart).

Il peut être nécessaire d'incrémenter une variable non pas de 1 en 1 mais d'une valeur (d'un pas) quelconque, il suffira de le préciser, ainsi :

```
10 FOR I = 1 TO 19 STEP 0.5
20 ...
30 NEXT I
```

La valeur I vaudra successivement 1, 1.5, 2, 2.5 etc.

Par ailleurs, on pourra également décrémenter une valeur, c'est-à-dire retirer une valeur précisée :

```
10 FOR I = 20 TO 1 STEP - 1
20 ...
30 NEXT
```

La variable I vaudra successivement 20, 19, 18, 17 etc.

5.4. NOTION DE PROGRAMMATION STRUCTUREE

WHILE WEND

Il est toujours préférable lors de l'écriture du programme de découper celui-ci (quand c'est possible, bien sûr) en de multiples petits morceaux indépendants les uns des autres ; cela permet de mieux le structurer afin de le rendre plus lisible pour une amélioration ou une correction éventuelle ; il n'y a rien de pire qu'un programme énorme rempli

de GOTO et mal structuré. On utilisera alors les instructions GOSUB et RETURN du paragraphe suivant mais également les instructions WHILE et WEND (de while, tant que et de WEND, revenir et en même temps jeu de mots sur While et END, fin de WHILE). Ces deux instructions marchent toujours par paire et s'apparentent à FOR...NEXT ; elles signifient que, TANT QU'un fait est vrai, le Basic analyse toutes les instructions comprises entre WHILE et WEND, par exemple :

```
10 INPUT A
20 WHILE A < 10
30    PRINT A: A = A+1
40 WEND
50 ...
```

Si la condition est fausse ou dès qu'elle n'est plus vérifiée alors le Basic continue l'analyse en ligne 50. De même que précédemment, on peut imbriquer des WHILE... WEND.

Ces deux instructions se distinguent des boucles en ce sens qu'elles agissent en fonction de l'état d'une valeur et non selon cette valeur. Ainsi, dans l'exemple, la ligne 10 vérifie que :

ou A est inférieur ou A n'est pas inférieur. Cette façon d'analyser peut remplacer le test (IF... THEN) : nous le verrons dans le chapitre consacré aux fichiers.

5.5. LES SOUS PROGRAMMES

GOSUB et RETURN

Plus nous avançons dans la programmation et plus nos programmes prennent de l'ampleur. Il devient alors important de pouvoir mieux les structurer pour mieux analyser et représenter un problème à résoudre.

Dans un programme il peut arriver qu'une même fonction soit utilisée de nombreuses fois ; ceci est également vrai pour un ensemble de fonctions, par exemple le calcul de factorielle que nous venons de voir : il est parfaitement concevable qu'un programme fasse appel plusieurs fois à ce type de calculs. Nous savons résoudre cela en écrivant la partie du programme correspondant. Or, il apparait évident, si par exemple nous devons dans un programme et pour une raison quelconque, faire 5 fois un calcul de factorielle, que cela serait du temps et de la place gaspillés que d'écrire 5 fois la MEME partie de pro-

gramme, à savoir ici le calcul de factorielle. Il serait éminemment souhaitable que de pouvoir écrire UNE seule fois cette partie et de l'utiliser CINQ fois. Or, nous le pouvons car nous avons appris l'instruction GOTO : il est donc possible de placer ce calcul de factorielle loin dans un programme et d'y accéder avec GOTO comme ceci :

```
10 GOTO 1000
20...
30...
50 GOTO 1000
60 ...
90 ...
.....
1000 REM CALCUL DE FACTORIELLE
1010 ....
```

Il demeure cependant un inconvénient de taille : si on sait COMMENT aller en ligne 1000, on ne sait pas COMMENT en revenir. Bien sûr, le premier réflexe est de dire : il suffit de mettre un GOTO quelconque APRES le calcul pour faire revenir le pointeur de là où il était parti. Or, ceci est vrai, certes, mais seulement dans le cas ou l'on veut faire revenir le pointeur TOUJOURS AU MEME ENDROIT : ainsi, dans l'exemple précédent pourrait-on écrire :

1100 GOTO 20

Quand le système lirait en ligne 10 d'aller en 1000, il le ferait bien sûr, puis lirait cette ligne 1100 qui lui dirait de se pointer en ligne 20, ce qu'il ferait aussi. Le problème se poserait à la ligne 50 ; le pointeur lisant le GOTO 1000 irait se positionner normalement en ligne 1000, comme on s'y attend puis lirait ensuite la ligne 1100 : or si dans notre esprit nous voulons que le système reparte en ligne 60, pour continuer l'exécution, la ligne 1100 obligerait le pointeur à aller toujours en ligne 20, ce qui ne nous arrange guère. Il faut donc utiliser une autre manière de procéder et nous le pouvons avec GOSUB et RETURN.

L'instruction GOSUB (de go subroutine, aller à un sous-programme) est absolument identique à GOTO. Pour aller à une partie située en ligne 1000 (nous appellerons plus précisément des parties de programme, des sous-programmes si ces parties sont indépendantes les une des autres), nous dirons simplement GOSUB 1000. La différence réside avec l'instruction RETURN (retour) qui obligera le pointeur à revenir LA OU IL ETAIT PARTI AVEC GOSUB, plus précisément à revenir à l'instruction suivant le GOSUB. Cette instruction,

RETURN, se placera à la fin du sous-programme. Ainsi, les problèmes précédemment évoqués disparaissent.

On peut facilement illustrer cette fonction. Dans une chanson, souvent il y a plusieurs couplets et un refrain. Si par exemple on veut écrire les paroles de cette chanson, comment procède-t-on ? On fait comme ceci :

Premier couplet .....
.....
.....

REFRAIN

Deuxième couplet

.....
.....
.....

REFRAIN

Troisième couplet

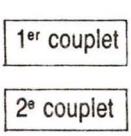
.....
.....
.....

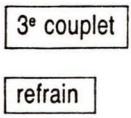
REFRAIN

Refrain .....
.....
.....

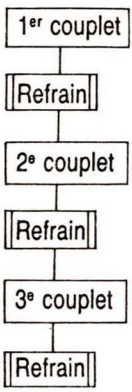
On utilise la notion de sous-programme en n'écrivant pas le refrain à chaque fin de couplet mais seulement le mot ''refrain'' qui le symbolise. Ce refrain est écrit une fois à la fin. Quand on chante la chanson en lisant les paroles, on chante d'abord le premier couplet puis, dès qu'on lit le mot ''refrain'', on se reporte en fin de chanson pour le chanter aussi ; quand on a fini ce refrain, on repart chanter LA SUITE ; cette suite est le second couplet qu'on chante ; on retrouve le mot ''refrain'' qui nous envoie à la fin et ainsi de suite. Ce mot ''refrain'' est en fait l'équivalent de GOSUB, un GOSUB qui renvoie en fin de chanson. Le RETURN est automatique, dans la chanson ! Si nous avions utilisé la notion de GOTO, nous ne retournions pas au couplet suivant, après chaque refrain, mais toujours au même couplet :

Organigramme de la chanson :

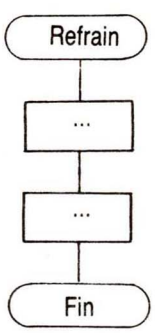




Le programme se dessine ainsi :



Le sous-programme se dessine ainsi :



Exercices

Ex. n° 1 :

Reprendre le jeu du nombre mystérieux et entrer dans le programme la capacité à afficher le nombre d'essais supérieurs, inférieurs et le nombre total d'essais.

Ex. n° 2 :

Ecrire le programme permettant de calculer la surface d'un rectangle connaissant la longueur et la largeur, ou bien de calculer la lon-

gueur connaissant la surface et la largeur, ou bien de calculer la largeur connaissant la surface et la longueur. On pourra écrire de deux façons le programme : une première sans utiliser GOSUB et RETURN, une seconde en les utilisant.

Ex. n° 3 :

Reprendre les tables de multiplication et faire afficher par le système les étapes Table 1, Table 2, etc...

Solutions

Ex. n° 1 :

```
10 REM -------------JEU DU NOMBRE MYSTERIEUX ----------
20 NM=INT(RND*100)
30 SUP=0: INF=0
40 INPUT "entrez le nombre proposé";NP
50 IF NM < NP THEN 110
60 IF NM > NP THEN 160
70 PRINT " c'est GAGNE en ";1+SUP+INF;" essais"
80 PRINT SUP; "SUPERIEURS"
90 PRINT INF; "INFERIEURS"
100 GOTO 20
110 REM -----------plus petit --------------------
120 REM
130 PRINT " plus petit "
140 INF=INF+1
150 GOTO 40
160 REM -----------plus grand --------------------
170 REM
180 PRINT "plus grand"
190 SUP=SUP+1
200 GOTO 40
```

On écrit en ligne 70 : 1 + SUP + INF ; le "1" sert à compter le dernier essai.

Ex. n° 2 :

— première façon :

```
10 REM -------------CALCULS SUR LE  RECTANGLE----------
20 PRINT "recherche  de"
30 PRINT "1  LA SURFACE"
40 PRINT "2  LA LONGUEUR"
```

```
50 PRINT "3  LA LARGEUR"
60 PRINT
70 INPUT  "entrez votre choix";c
80 IF C=1 THEN 120
90 IF C=2 THEN 200
100 IF C=3 THEN 260
110 GOTO 10
120 REM
130 REM --------------SURFACE ------------------------
140 REM
150 INPUT "entrez la longueur ";LO
160 INPUT "entrez la largeur ";LA
170 PRINT "surface=";LO*LA
180 GOTO 10
190 REM
200 REM ------------LONGUEUR -------------------------
210 REM
220 INPUT "entrez la surface ";S
230 INPUT "entrez la largeur ";LA
240 PRINT "longueur=";S/LA
250 GOTO 10
260 REM
270 REM -----------LARGEUR ---------------------------
280 REM
290 INPUT "entrez la surface ";S
300 INPUT "entrez la longueur ";LO
310 PRINT "largeur=";S/LO
320 GOTO 10
```

— seconde façon :

```
10 REM -------------CALCULS SUR LE  RECTANGLE----------
20 PRINT "recherche  de"
30 PRINT "1  LA SURFACE"
40 PRINT "2  LA LONGUEUR"
50 PRINT "3  LA LARGEUR"
60 PRINT
70 INPUT  "entrez votre choix";c
80 IF C=1 THEN GOSUB 120
90 IF C=2 THEN GOSUB 200
100 IF C=3 THEN GOSUB  260
110 GOTO 10
120 REM
130 REM --------------SURFACE ------------------------
140 REM
```

```
150 GOSUB 330
160 GOSUB 350
170 PRINT "surface=";LO*LA
180 RETURN
190 REM
200 REM ------------LONGUEUR -------------------------
210 REM
220 GOSUB 370
230 GOSUB 350
240 PRINT "longueur=";S/LA
250 RETURN
260 REM
270 REM -----------LARGEUR ---------------------------
280 REM
290 GOSUB 370
300 GOSUB 330
310 PRINT "largeur=";S/LO
320 RETURN
330 INPUT "entrez la longueur ";LO
340 RETURN
350 INPUT "entrez la largeur ";LA
360 RETURN
370 INPUT "entrez la surface ";S
380 RETURN
```

Commentaires

Même si le second programme est plus long que le premier, il est néanmoins plus structuré et mieux construit.

Ex. n° 3 :

```
10 REM ---------------TABLES----------------------------
20 FOR J=1 TO 10
30    PRINT "table de ";J
40    PRINT
50       FOR I=1 TO 10
60         PRINT  J;" * ";I; " = ";J*I
70       NEXT
80    PRINT
90 NEXT
```

Rien de bien compliqué ici.

6.
LE STOCKAGE ET LA MANIPULATION DES INFORMATIONS

6.1. EN MEMOIRE VIVE

DIM, ERASE

6.1.1. Tableau à une dimension

6.1.1.1. Tableau numérique

Pour mettre en mémoire des valeurs, que ce soit des constantes ou des variables, alphanumériques ou numériques, nous avons utilisé jusqu'à présent des noms particuliers les désignant. Seul le signe dollar, ''$'', placé après le nom précisait la qualité alphanumérique de la valeur en question. Par exemple, si l'on veut représenter le prix de cinq produits, il suffit de dire :

```
10 PRIX1 = 123.5
20 PRIX2 = 34.6
30 PRIX3 = 12.4
40 PRIX4 = 100
50 PRIX5 = 120
60 .....
```

Pour effectuer par la suite un calcul quelconque utilisant de telles valeurs, il suffit de rappeler les valeurs en question par leur nom. Ainsi, connaître le prix total ne pose aucun problème :

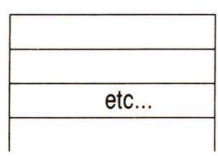
TOTAL = PRIX1 + PRIX2 + PRIX3 + PRIX4 + PRIX5

L'inconvénient d'une telle méthode de représentation des données apparaît dès que l'on veut en manipuler beaucoup. Pour 200 prix, il faudrait écrire 200 fois ''PRIXnuméro de la valeur donnée = etc''. De plus, des problèmes énormes se posent pour manipuler toutes ces données qui ont un nom différent, et pour cause !

Les concepteurs du Basic ont eu pitié des pauvres programmeurs et ont prévu une instruction facilitant leur travail. Ils ont créé une nouvelle notion, celle du tableau de données.

Un tableau est un ensemble de cases-mémoire que l'utilisateur remplit à son gré et que l'on peut imaginer ainsi :

tableau



Un tableau doit porter un nom qui l'identifie car on peut en créer autant que l'on veut (dans les limites de la mémoire disponible) et chaque case du tableau porte le même nom que lui. Pour distinguer cependant la case n° 6 de la case n° 10, on adjoint à cette case son... numéro d'emplacement ; ainsi, si le tableau se nomme PRIX, la case n° 6 se nommera PRIX(6) et la case n° 10 se nommera PRIX(10) :

tableau PRIX

PRIX (1)	
PRIX (2)	
PRIX (3)	
....	
PRIX (6)	
....	
PRIX (10)	
....	

Pour utiliser un tel tableau il faut avant toute action le déclarer avec l'instruction DIM PRIX(200), qui dimensionne en mémoire un

tableau nommé PRIX comportant 200 cases. Il est obligatoire de déclarer le tableau avant son utilisation (sinon le système renvoie le message ''subscript out of range'') car le Basic réserve dans la mémoire disponible les 200 cases en question.

L'utilisation d'un tableau, et plus précisément de ses cases-mémoire se fait exactement de la même manière qu'une valeur normale. Pour entrer 1123 dans la troisième case, on peut le faire ainsi :

```
INPUT PRIX(3), puis on entre 1123, ou bien ainsi :
PRIX(3) = 1123
```

Pour calculer, on fait exactement pareil qu'avant :

PRIX(3) = PRIX(1) * PRIX(2). Ici, le nombre contenu dans la première case est multiplié par le contenu de la deuxième et le résultat est placé dans la troisième.

Comment utiliser les tableaux, car après tout cela ne change apparemment pas grand-chose. En fait, et on s'en doute un peu, c'est surtout la conjonction de la fonction FOR.. NEXT et des tableaux qui multiplie les potentialités.

Il est vrai que par exemple, pour entrer les 200 valeurs, on peut procéder comme ceci :

```
10 DIM PRIX(200)
20 INPUT PRIX(1)
30 INPUT PRIX(2)
40 INPUT PRIX(3)    et ceci 200 fois... On n'est pas
                    vraiment avancé
```

Devant une telle manipulation de données, on doit acquérir le réflexe de la boucle, puisque visiblement une valeur s'incrémente de 1 en 1, la valeur définissant l'indice de la case-mémoire ; or, seule la boucle, donc FOR... NEXT, peut le faire :

```
10 DIM PRIX(200)
20 FOR I = 1 TO 200
30      INPUT "entrez le prix "; PRIX( I )
40 NEXT
```

Dans ce programme, on comprend que : quand I vaut 1, PRIX (I) signifie PRIX (1), puis quand I vaut 2, PRIX(i) signifie PRIX(2) etc. Ainsi, c'est comme si on écrivait 200 fois ''INPUT PRIX (quelque chose), à la différence près que maintenant c'est l'ordinateur qui s'en charge.

On comprend maintenant l'intérêt et des tableaux et des boucles : les deux vont souvent ensemble.

Ainsi, pour relire les valeurs précédemment entrées, il suffira tout simplement de refaire une boucle... :

```
50 FOR I = 1 TO 200
60   PRINT  PRIX ( I )
70 NEXT
```

(petit conseil : si vous faites des essais, n'entrez pas 200, mais 3, par exemple...)

IMPORTANT

Il peut arriver que l'on oublie de déclarer le tableau PRIX, (par exemple) mais de vouloir néanmoins l'utiliser. Le Basic le créera automatiquement dès qu'il rencontrera l'expression ''INPUT PRIX (I) ''ou'' PRINT PRIX(I). Mais le tableau ainsi créé sera dimensionné seulement sur dix cases : c'est comme si on avait dit ''DIM PRIX(10)''. Du coup, si le programme utilise un indice supérieur à 10, le système répond ''SUBSCRIPT OUT OF RANGE'', soit ''Donnée hors du rang, dépassement de la capacité du tableau''.

Nous considérons pour notre part qu'il s'agit ici d'une erreur de conception, et susceptible de gêner surtout les débutants !

Voyons quelques applications.

Pour recopier un tableau dans un autre :

Soit un tableau A rempli de valeurs diverses et un tableau B vide :

```
10 DIM A(5), B(5)    ← dimensionnement des deux tableaux
20 FOR I=1 TO 5
30   INPUT A(I)      ← entrée des valeurs
40 NEXT
50 FOR I=1 TO 5
60   B(I)=A(I)       ← copie de A(I) en B(i)
70 NEXT
100 FOR I=1 TO 5
110  PRINT B(I)      ← lecture du tableau B
120 NEXT
```

Il faut bien voir ici que les valeurs du tableau A restent toujours dans ce tableau ; elles sont seulement dupliquées en B.

Pour faire un total de valeurs :

Reprenons l'exemple du début ; après avoir entré une série de prix, on désire en faire la somme :

```
10 DIM PRIX(5)
20 FOR I= 1 TO 5
30  INPUT PRIX(I)
40 NEXT
```

Le tableau PRIX est maintenant ainsi :

PRIX (1)	123.5
PRIX (2)	34.6
PRIX (3)	12.4
PRIX (4)	100
PRIX (5)	120

Pour faire la somme, on pourrait simplement écrire :

TOTAL = PRIX(1) + PRIX(2) + PRIX(3) + PRIX(4) + PRIX (5). Ceci est bien sûr bon. Cependant, on voit de nouveau le problème se poser si nous additionnons 200 nombres ; il faut donc procéder autrement.

Pour faire la somme totale, on va additionner le 1er nombre au second, puis additionner ce total partiel, 1er + 2e au troisième nombre, puis additionner ce total partiel, (1er + 2e) + 3e, au quatrième puis additionner ce total partiel, ((1er + 2e) + 3e) + 4e, au cinquième :

```
TOTAL =  ( ( ( 1er + 2ème ) + 3ème ) + 4ème ) + 5ème.
```

On voit qu'ici, une même opération se répète : il suffit de compter le nombre de "additionner" dans la phrase précédente. Or, on sait maintenant faire répéter une quelconque action en Basic avec l'instruction FOR... TO... NEXT. Nous allons donc utiliser une boucle pour faire se répéter le total partiel dont nous parlions précédemment.

Ouvrons ici une parenthèse ; quand nous avons étudié la notion de compteur, que faisions-nous ? On incrémentait une valeur, le compteur, de 1 à chaque passage. Or, ici, qu'avons-nous avec cette notion de total partiel ? Exactement la même chose, à ceci près qu'au lieu d'additionner la même valeur 1, on additionne une valeur différente, PRIX(I) ; et cette valeur PRIX(I), nous l'additionnons à une variable TOTAL, qui va donc symboliser à chaque passage le cumul partiel des

nombres. Ainsi, en faisant la conjonction de la notion de compteur et celle de boucle, nous obtenons :

```
100 TOTAL = PRIX(1)
110  FOR I = 2 TO 5
120      TOTAL = TOTAL + PRIX(I)
130 NEXT
140 PRINT "total = ";TOTAL
```

Dans ce programme, on crée au départ une variable nommée TOTAL dans laquelle on place la première valeur du tableau, car il faut bien commencer quelque part. On fait démarrer la boucle à 2 pour ne pas additionner la première valeur à elle-même. En plaçant en ligne 115

PRINT TOTAL

Nous pourrons lire le cumul partiel.

Petit exercice : peut-on écrire le programme autrement ? Oui, en supposant au départ la valeur TOTAL égale à zéro :

```
100 TOTAL = 0
110 FOR I = 1 TO 5
120   TOTAL = TOTAL + PRIX(I)
130 NEXT
140 PRINT TOTAL
```

6.1.1.2. Tableaux alphanumériques

Nous n'avons parlé jusqu'à présent que de valeurs numériques dans les tableaux. Or, nous pouvons parfaitement utiliser les tableaux pour y stocker des valeurs alphanumériques. Nous savons déjà que le Basic distingue ces deux types de valeurs grâce au signe dollar, $ placé après le nom de la valeur. Ainsi, pour créer un tableau dans lequel nous placerons des chaînes de caractères, nous écrirons simplement :

DIM NOM$(100)

pour déclarer un tableau nommé NOM et composé de 100 cases. Bien sûr, il ne sera pas question de faire des opérations arithmétiques avec un tel tableau, mais tout ce qui a été dit précédemment concernant les tableaux numériques reste valable pour les tableaux alphanumériques.

Problème :

Voici une classe d'élèves connus par leur prénom, âge, poids et taille. Comment les représenter sur l'écran ou l'imprimante en indiquant de plus l'âge moyen, le poids moyen et la taille moyenne ?

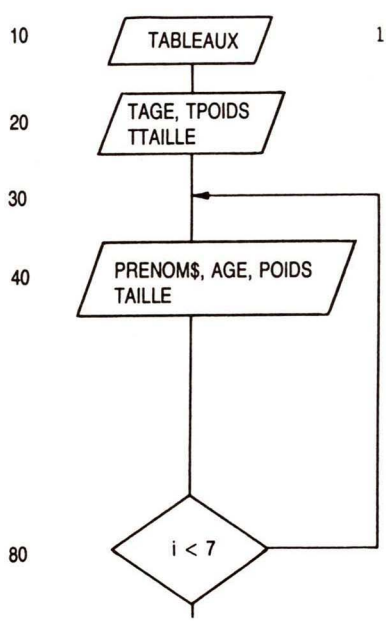
Prénoms	Age	Poids	Taille
Patrice	13	48	130
Denise	14	56	123
Pierre	13	55	129
Annick	12	50	133
François	13	49	125
Marie	12	52	122

Algorithme :

Il y a quatre séries de valeurs, dont une alphanumérique, on peut donc prévoir quatre tableaux, de 6 cases chacun : PRENOM$(6), AGE(6), POIDS(6) et TAILLE(6) ; de plus l'entrée de données peut être facilitée par l'usage de boucles.

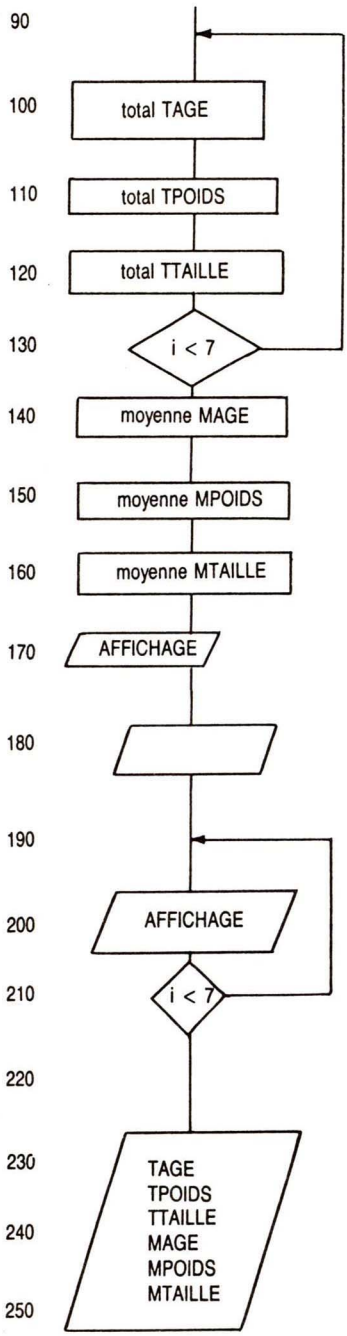
Les calculs à effectuer, en portant sur des moyennes, impliquent l'usage de totaux : il y aura donc des valeurs symbolisant le total des âges, des poids et des tailles : TAGE, TPOIDS et TTAILLE, ainsi que les moyennes correspondantes : MAGE, MPOIDS et MTAILLE.

Organigramme



Programme

```
10 DIM PRENOM$(6),AGE(6),POIDS(6),TAILLE(6)
20 TAGE=0:TPOIDS=0:TTAILLE=0
30 FOR I = 1  TO 6
40 INPUT "prénom";PRENOM$(I)
50 INPUT "age ";AGE(I)
60 INPUT "poids"; POIDS(I)
70 INPUT "taille ";TAILLE(I)
80 NEXT
```

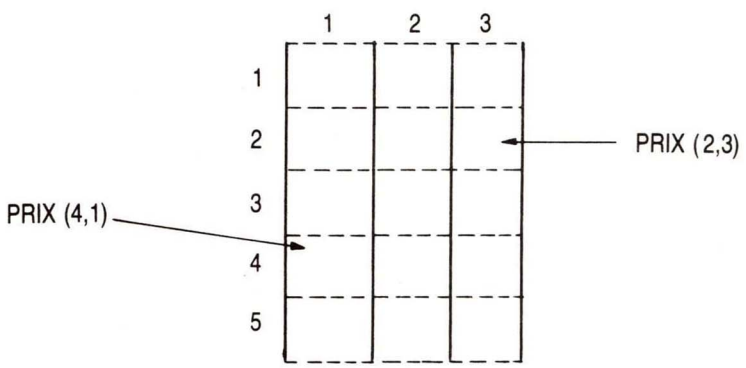



```
 90 FOR I = 1 TO 6
100 TAGE= TAGE + AGE(I)
110 TPOIDS= TPOIDS + POIDS(I)
120 TTAILLE= TTAILLE + TAILLE(I)
130 NEXT
140 MAGE = TAGE / 6
150 MPOIDS= TPOIDS / 6
160 MTAILLE= TTAILLE / 6
170 PRINT "* Prénom";TAB(20);"age";TAB(40)
    ;"poids";TAB(60);"taille *"
180 PRINT
190 FOR I = 1 TO 6
200 PRINT PRENOM$(I);TAB(20);AGE(I);TAB(40
    ;POIDS(I);TAB(60);TAILLE(I)
210 NEXT
220 PRINT
230 PRINT "total";TAB(20);TAGE;TAB(40);
    TPOIDS;TAB(60);TTAILLE
240 PRINT
250 PRINT "moyenne";TAB(20);MAGE;TAB(40);
    ;MPOIDS;TAB(60);MTAILLE
```

6.1.2. Tableaux à deux dimensions

Nous venons de voir des tableaux possédant une seule colonne. On les appelle "tableau à une dimension". Or, il est possible de créer des tableaux ayant plusieurs colonnes : nous les nommerons "tableaux à deux dimensions". Ils se présentent et s'utilisent comme les précédents, à la différence près que l'on doit préciser le nombre de colonnes lors de leur déclaration. Par exemple, si on désire créer un tableau nommé PRIX et ayant 5 lignes et 3 colonnes, nous dirons :

DIM PRIX (5, 3) : on déclare toujours le nombre de lignes précédant le nombre de colonnes.



Il va sans dire que de tels tableaux sont obligatoirement ou numériques, ou alphanumériques, mais jamais un mélange des deux.

Pour illustrer ces tableaux, reprenons l'exemple précédent. Au lieu de déclarer quatre tableaux, nous allons seulement en déclarer deux : le premier sera toujours PRENOM$; le second sera un tableau numérique à deux dimensions : il aura 8 lignes et 3 colonnes réservées à l'âge, le poids et la taille et nous le déclarerons ainsi :

DIM PRENOM$(8), APT(8,3). Pourquoi 8 lignes ? Simplement pour inclure maintenant dans le tableau alphanumérique les mots "total" et "moyenne" et pour inclure dans APT les valeurs calculées des totaux et des moyennes.

Chaque case est maintenant repérée par ses coordonnées, numéro de ligne et numéro de colonne, comme à la bataille navale. Pour entrer les valeurs de l'âge, du poids et de la taille, nous écrirons ainsi :

```
10 DIM PRENOM$(8),APT(8,3)
20 FOR I=1 TO 8
30    INPUT  "entrez le prénom ";PRENOM$(I)
40    INPUT  "entrez l'âge ";APT(I,1)
50    INPUT  "entrez le poids";APT(I,2)
60    INPUT  "entrez la taille";APT(I,3)
70 NEXT
80  FOR I= 1 TO 6
90    APT(7,1)=APT(7,1) + APT(I,1)
100   APT(7,2)=APT(7,2) + APT(I,2)
110   APT(7,3)=APT(7,3) + APT(I,3)
120 NEXT
130 APT(8,1)= APT(7,1) / 6
140 APT(8,2)= APT(7,2) / 6
150 APT(8,3)= APT(7,3) / 6
160 FOR I=1 TO 8
170 PRINT "prenom";TAB(20);"age";TAB(40);"poids";TAB
        (60);"taille"
180 PRINT
190 FOR I=1 TO 8
200  PRINT PRENOM$(I);TAB(20);APT(I,1);TAB(40);APT
        (I,2);TAB(60);APT(I,3)
210 NEXT
```

Commentaires :

La ligne 20 indique une boucle allant jusqu'à 8 ; nous entrerons donc le nom des 6 personnes puis les mots "total" et "moyenne" à la question "entrez le prénom" et nous frapperons RETURN quand nous aurons épuisé les valeurs. On notera que les totaux et les moyennes sont inclus dans le tableau.

ERASE

On peut effacer un tableau avec l'ordre ERASE (de erase, effacer) ; on l'utilise ainsi, et dans le cas ou l'on voudrait effacer les tableaux A, B et C$:

```
ERASE A,B, C$
```

6.2. DANS LE PROGRAMME

DATA, READ et RESTORE

On peut stocker des valeurs dans le programme lui-même avec l'instruction DATA (de data, donnée). Ainsi, différentes valeurs numériques ou alphanumériques peuvent s'écrire :

DATA Jean, 18, Paul, 20, Arthur, 19 etc.

Ce sont les virgules qui déterminent les valeurs. Il faut bien noter que ce sont des valeurs, c'est-à-dire des contenus éventuels de variables qu'il faut cependant créer. Une ligne de DATA ne sert à rien si elle ne correspond pas à des variables existantes.

Pour lire ces valeurs, il faut donc les entrer dans une variable ou des variables avec l'ordre READ (de read, lire) ; cette instruction lit chaque donnée séquentiellement, c'est-à-dire l'une derrière l'autre. Soit le programme suivant :

```
10 DATA 12, 130, 23, 34,45
20 READ C
30 PRINT C
```

Si nous lançons RUN, le programme lit en 20 la première valeur des DATA, soit 12, et place automatiquement 12 dans C : c'est exactement comme si la ligne 20 aurait été :

20 C = 12

ou bien

20 INPUT C et que l'on aurait entré 12.

Plus intéressant est le programme suivant qui lit et donc peut utiliser des valeurs spécifiées :

```
10 DATA 12, 130, 23, 34, 45
20 FOR I= 1 TO 5
30 READ C
40 PRINT C
50 NEXT
```

Ici, chaque passage en ligne 30 oblige le pointeur à aller chercher chaque valeur en DATA successivement ; il faut bien comprendre que le Basic a un compteur interne de DATA : dès qu'il lit un DATA, le troisième par exemple, il sait que le DATA suivant à lire, si on lui demande, est le quatrième. Dans notre exemple, la variable C vaudra successivement toutes les valeurs contenues dans la ligne DATA. Il est à noter que si le programme demande de lire un sixième DATA, le système répondra :

DATA exhausted, soit : il n'y a plus de DATA à lire.

De plus, si la ligne de DATA est plus longue que prévue, rien n'empêche d'écrire une autre ligne de DATA à la suite. Il faut savoir

que l'on peut placer les lignes de DATA n'importe où dans la programme. Le pointeur, à la recherche de DATA, se positionnera automatiquement à la ligne de DATA suivante, même si celle-ci se trouve 3000 lignes plus loin. Il peut être d'ailleurs intéressant de créer plusieurs lignes DATA car ceci nous permet de pouvoir gérer ces lignes plus finement qu'il n'y paraît avec l'instruction RESTORE (restaurer, remettre). En effet, il peut s'avérer utile d'utiliser une ligne de DATA plutôt qu'une autre et pour spécifier cela on précisera, avant la lecture par READ :

RESTORE n° de la ligne des DATA en question. Ainsi :

```
10 DIM DONNEE(5)
20 DATA  6, 8, 3, 4, 7
30 DATA  2, 7, 6, 9, 0
40 RESTORE 30
50  FOR I = 1 TO 5
60    READ DONNEE(I)
70  NEXT
```

Dans cet exemple, la ligne de DATA n° 20 est totalement ignorée par le programme car l'ordre RESTORE oblige à utiliser en ligne 60 les DATA de la ligne 30. Rien n'empêche cependant d'user de la ligne 20 ultérieurement, il faudra alors écrire :

RESTORE 20

Quel est l'intérêt de ces instructions - Data, Read, Restore - ?

L'intérêt est de deux ordres. On peut avoir besoin d'utiliser des données constantes dans un programme : cela arrive même souvent. Pour ce faire, on se servait jusqu'à présent de la notion de constantes simples, c'est-à-dire de déclarer en début de programme les valeurs en question, et de les écrire les unes derrière les autres. Avec l'usage des DATA, rien ne change de façon formelle, seulement une manière plus rationnelle et élégante de programmer. Supposons avoir besoin, dans un quelconque programme, de 30 valeurs constantes. Plutôt que de les déclarer comme on le faisait jusqu'à présent en leur donnant un nom puis en les écrivant dans le programme, on peut placer ces constantes dans un tableau et les faire entrer automatiquement dans ce tableau avec l'instruction READ ; ce qui nous donne :

```
10  DIM DONNEE(30)
20  DATA 1, 6, 6, 7, 3, 4, 6, 7, 8 ,9, 5, 4, 5, 6 ,7,
      8, 9, 0 ,9, 8, 7, 6 ,5 ,5, 4, 3, 4, 5, 6, 7
30   FOR I = 1 TO 30
40      READ DONNEE(I)
50   NEXT
```

Il va sans dire que les Data peuvent être également alphanumériques.

Par ailleurs, on peut avoir intérêt à utiliser les DATA pour s'aider dans l'écriture d'un programme. Bien souvent, le programme que l'on est en train d'écrire nécessite de nombreuses mises au point ; bien souvent, il faut entrer des valeurs pour tester le programme. Or, plutôt que d'entrer à chaque essai ces valeurs, on aura intérêt à les mettre en DATA et à les faire entrer automatiquement à l'aide d'une boucle. Puis, une fois le programme au point, ôter ces lignes de DATA.

6.3. TRI ET SELECTION DE DONNEES :

SWAP MAX MIN

6.3.1. Recherche du plus grand ou du plus petit élément d'une liste

On a souvent besoin de classer, trier ou sélectionner des données selon un certain ordre ; par exemple, on peut vouloir la plus grande valeur ou la plus petite d'une série de nombres. Soit la série suivante :

3, 4, 5, 1, 7, 6, 9, 10, 0, 4 composée de 10 nombres. Pour l'analyser on peut la mettre en DATA puis la stocker en un tableau A(10) (à noter que dans cet exemple il n'est pas nécessaire d'utiliser les tableaux : on le fera cependant en prévision de la suite) :

```
10 DIM A(10)
20 DATA 3, 4, 5, 1, 7, 6, 9, 10, 0, 4
30 FOR I= 1 TO 10:READ A(I) : NEXT
```

Pour chercher dans un tableau la plus petite valeur, on peut se dire qu'il suffit de prendre la première, puis de la comparer à la seconde et de garder la plus petite des deux ; ensuite de comparer cette dernière à la troisième valeur et de garder également la plus petite et de continuer jusqu'à la dernière valeur du tableau pour obtenir ainsi la plus petite de toutes les valeurs. Comme on doit avoir une variable servant à la comparaison et qui stockera la plus petite valeur à chaque étape d'une part, et d'autre part comme nous postulons une répétition de traitement, on utilisera une boucle :

```
40 PP = A(1)
50 FOR I=2 TO 10
60    IF A(I) < PP THEN PP= A(I)
```

```
70 NEXT
80 PRINT "plus petite valeur ";PP
```

On voit qu'au départ, PP vaut la première valeur, car il faut bien commencer. En ligne 60, la comparaison porte sur cette valeur PP et chaque valeur du tableau : si la valeur du tableau est plus petite que la valeur supposée l'être, PP, on permute alors et PP vaut A(I), et si la valeur du tableau est plus grande que PP, il ne se passe rien, ce qui fait qu'au sortir du test, PP dans tous les cas sera la plus petite valeur analysée.

On comprend qu'en utilisant l'opérateur " > " en ligne 60, nous obtiendrons la plus grande valeur du tableau.

Les concepteurs du Basic de l'AMSTRAD ne reculant devant rien, l'ont doté de deux fonctions faisant rigoureusement la même chose, MAX et MIN, qui signifient bien sûr maximum et minimum. On les utilise ainsi :

PRINT MAX (1, 5, 9, 2, 3) et nous obtenons le plus grand, 9 et :

PRINT MIN (1, 5, 9, 2, 3) et nous obtenons le plus petit, 1.

Pour utiliser ces deux fonctions en analysant un tableau comme dans l'exemple précédent, on peut changer la ligne 60 ainsi :

60 PP = MIN (PP, A(I))

ou bien :

60 PP = MAX (PP, A(I)) et nous aurons le plus petit ou le plus grand nombre du tableau.

6.3.2. Le tri de données :

Afficher par ordre croissant le tableau précédent, c'est chercher le plus petit, l'écrire en premier puis le gommer et c'est ensuite refaire de même avec toutes les valeurs du tableau. Comment le faire en Basic ? On peut prendre pour base qu'il faut analyser chaque valeur, donc faire une boucle J, par exemple ; puis qu'il faut comparer chaque valeur d'indice J avec chacune des autres valeurs, donc refaire une autre boucle, I par exemple ; si la comparaison est bonne, c'est-à-dire avoir trouvé une valeur inférieure, on stockera cette valeur dans un autre tableau qui servira de tableau trié. On prendra soin de gommer du tableau initial la valeur trouvée pour qu'elle ne soit plus prise en compte, en la remplaçant par la plus grande valeur possible :

```
10 DIM A(10), B(10)
20 DATA 3, 4, 5, 1, 7, 6, 9, 10, 20, 4
```

```
30 FOR I= 1 TO 10: READ A(I): NEXT
40 FOR J=1 TO 10
50    B(J)= A(J)
60      FOR I=1 TO 10
70        IF A(I) <= B(J) THEN  B(J)=A(I):LIEU=I
80      NEXT I
90    A(LIEU)=1E+38
100 NEXT J
110 FOR I=1 TO 10:PRINT B(I);" ";:NEXT
```

On remarquera la ligne 50, qui permet de donner une valeur à la variable de comparaison et les lignes 70 et 90. La valeur LIEU sert à mémoriser l'indice du tableau A où se trouve la valeur à gommer et cette dernière est gommée en ligne 90. Plus précisément on annihile cette valeur en la remplaçant par la plus grande valeur possible, un 1 suivi de 38 zéros, écrits ici en notation scientifique. Il va sans dire que ce programme peut trier par ordre décroissant, il suffit d'inverser le signe "<" en ">" et la ligne 90 en 1E-38, qui est la plus petite valeur possible.

Le tri par permutation

On peut trier en réalité de plusieurs manières qui se distinguent les unes des autres par leur complexité et donc leur vitesse. Mais une constante demeure : comment permuter deux valeurs entre elles ? Supposons avoir les deux valeurs suivantes :

A = 5
B = 3

Nous voulons obtenir :

A = 3
B = 5

On ne peut simplement dire : A = B et B = A, car ce faisant, nous obtiendrons A = 3 et B = 3. En fait, pensons d'une manière plus triviale à un autre problème où la solution saute aux yeux (attention tout de même !). Si l'on possède un verre de vin et un verre de bière et que nous voulons mettre le vin dans le verre à bière et la bière dans le verre à vin, on voit immédiatement que la seule solution consiste à utiliser un troisième récipient ; il en est de même avec nos deux valeurs A et B ; nous allons donc créer une troisième valeur, C, et nous pouvons donc dire :

C = A
A = B
B = C

Ainsi : C = 3, A = 5 et B = 3 : nous avons bien permuté les deux valeurs A et B. Mais les concepteurs du Basic, dans leur bienveillante mansuétude, ont voulu nous épargner même cela : ils ont donc inventé l'instruction SWAP (de swap, échange), qui permute à notre place deux valeurs :

SWAP A,B est pareil à C = A:A = B:B = C.

Considérons maintenant un tableau quelconque, par exemple celui du programme précédent ; on peut l'imaginer ainsi :

Tableau A(10)

Indices	Valeurs
1	3
2	4
3	5
4	1
5	7
6	6
7	9
8	10
9	20
10	4

Quand on a I = 7, A(I) = A(7) donc A(I) = 9

Quand on a I = 7, A(I + 1) = A(8), donc A(I + 1) = 10. A ne pas confondre bien sûr avec A(I) + 1 qui serait A(7) + 1, soit 9 + 1, donc 10. Lumineux, non ?

Pour revenir au tri, on peut donc dire que trier une liste de valeurs c'est permuter une valeur et la suivante si cette dernière est plus petite que la précédente. Nous aurons le programme suivant :

```
10 DIM A(5)
20 DATA 3, 4, 5, 1, 7
30 FOR I=1 TO 5: READ A(I): NEXT
40 FOR J=1 TO 5
50     FOR I=1 TO 4
60         IF A(I) => A(I+1) THEN  SWAP A(I),A(I+1)
70     NEXT
80 NEXT
90 FOR I=1 TO 5: PRINT A(I);" ";:NEXT
```

On notera la ligne 50 dont la boucle s'arrête à 4, alors qu'il y a 5 valeurs ; cela s'explique par le fait que le test de la ligne 60 porte sur A (I + 1) : quand I vaudra 5, A(I + 1) = A(6), or il n'y a pas de sixième valeur : cela provoquera une erreur. Si l'on place en ligne 35 la ligne 90, nous pouvons voir les étapes intermédiaires :

1ere étape : 3 4 5 1 7
2e " 3 4 1 5 7
3e " 3 1 4 5 7
4e " 1 3 4 5 7
5e " 1 3 4 5 7

Il est évident par ailleurs que l'on peut trier des valeurs alphanumériques ; l'ordre croissant ou décroissant portera sur l'ordre alphabétique, étant bien entendu que le Basic connaît cet ordre et sait que la lettre A est plus "petite", donc vient avant, que la lettre B et ainsi de suite pour tous les caractères de l'alphabet.

Exercices

Ex. n° 1 :

Voici la liste des départements de la Champagne-Ardennes avec leur surface, leur population. Faites afficher sur l'écran ou l'imprimante un tableau résumant ces valeurs et précisant de plus la densité de la population par département ainsi que le pourcentage des populations et des superficies par rapport à l'ensemble. L'affichage devra être clair et les valeurs numériques alignées ; l'ensemble devra être agrémenté d'astérisques pour faire joli !

Ardennes 5253 km² 302 338 h ;
Aube 6002 km² 289 300 h ;
Marne 8196 km² 543 627 h ;
Haute-Marne 6250 km² 210 670 h ;

Ex. n° 2 :

Faire le programme permettant de gérer un fichier téléphonique composé des rubriques suivantes : nom, prénom, adresse, ville, téléphone. Le programme offrira les possibilités suivantes : ajout de nouvelles personnes, effacement de personnes, lecture séquentielle de tout le fichier, recherche d'après le nom, d'après le prénom, d'après la ville et d'après le n° de téléphone.

Solutions

Ex. n°1 :

Algorithme

Il sera nécessaire de faire un tableau pour les départements et un autre, numérique pour les valeurs données ; comme il y a 5 colonnes (pour la superficie, la population, la densité, le pourcentage de population et le pourcentage de superficie) et au moins 6 lignes, (quatre départements + une ligne réservée au total et une autre à la moyenne), nous pourrons déclarer un tableau alphanumérique DEP$(6) et un autre alphanumérique DEP(6,5) ; nous pourrions tout aussi bien déclarer cinq tableaux numériques pour chaque série de nombres.

Le tableau DEP(6,5) comprendra donc des constantes, données dans l'énoncé et des variables issues des calculs suivants :

Pour la densité : on divise la population par la superficie considérée ;

Pour les pourcentages : on divise la valeur considérée par le total de la série analysée et on multiplie par 100.

L'ordre à suivre sera :

1 — Entrée des données
2 — Calcul des totaux (car il faut connaître ces valeurs pour faire les calculs de pourcentages)
3 — Calcul des moyennes
4 — Calcul des densités et pourcentages
5 — Affichage des résultats

```
10   DATA   ARDENNES,5253,302338,   AUBE,6002,   289300,MARNE   ,8196,543627,
HAUTE/MARNE ,6250,210670 ,TOTAL,0,0 ,moyenne,0,0
20 DIM DEP$(6),dep(6,5)
30 F$="* ######      *     #######     *     ##.##     *     ##.##     *     ###.##
*"
40 TRAIT$=STRING$(83,"*")
50 FOR I=1 TO 6
60 READ DEP$(i)
70 READ DEP(i,1)
80 READ DEP(i,2)
90 NEXT
100 FOR i=1 TO 4
110 DEP(5,1)=DEP(5,1)+dep(i,1)
120 DEP(5,2)=DEP(5,2)+dep(i,2)
130 NEXT
140 DEP(6,1)=dep(5,1)/4
150 DEP(6,2)=dep(5,2)/4
160 FOR I=1 TO 4
170 DEP(i,3)=DEP(i,2)/DEP(i,1)
180 DEP(i,4)=DEP(i,1)/DEP(5,1) * 100
190 DEP(i,5)=DEP(i,2)/DEP(5,2) * 100
```

```
200 NEXT
210 LPRINT TRAIT$
220 LPRINT "département";tab(15);"superficie";TAB(30);"population";
TAB(45);"densité";TAB(60);"%SUP";TAB(75);"%POP"
230 LPRINT TRAIT$
240 FOR I=1 TO 6
250 LPRINT DEP$(i);TAB(15) USING F$;  dep(i,1);dep(i,2);dep(i,3);
dep(i,4);dep(i,5)
260 IF I = 4 THEN LPRINT TRAIT$
270 NEXT
280 LPRINT TRAIT$
```

Commentaires et résultats :

Les données sont entrées en DATA. On aurait pu mettre INPUT à la place de READ.

```
**************************************************************************
Département    superficie     population     densité    %SUP      %POP
**************************************************************************
ARDENNES     *  5253        *  302338      *    57.56 *  20.44  *  22.45
AUBE         *  6002        *  289300      *    48.20 *  23.35  *  21.49
HAUTE/MARNE  *  8916        *  543627      *    66.33 *  31.89  *  40.39
MARNE        *  6250        *  210670      *    33.71 *  24.32  *  15.65
**************************************************************************
total        * 25701        * 1345935      *     0.00 *   0.00  *   0.00
moyenne      *  6425        *  336483      *     0.00 *   0.00  *   0.00
**************************************************************************
```

Ex. n°2 :

```
10 REM
20 N=10
30 DIM n$(n),p$(n),a$(n),v$(n),t$(n)
40 PRINT
50 PRINT "                1 - CREATION DU FICHIER
60 PRINT "                2 - RECHERCHE D APRES LE NOM
70 PRINT "                3 - RECHERCHE D APRES LE TELEPHONE
80 PRINT "                4 - RECHERCHE D APRES LA VILLE
90 PRINT "                5 - EFFACEMENT D UN NOM
100 PRINT "                6 - ENTREE D UN NOM
110 PRINT "                7 - LECTURE DU FICHIER
120 INPUT " votre choix ";C
130 IF C=1 THEN GOSUB 210
140 IF C=2 THEN GOSUB 320
150 IF C=3 THEN GOSUB 410
160 IF C=4 THEN GOSUB 490
170 IF C=5 THEN GOSUB 570
180 IF C=6 THEN GOSUB 660
190 IF C=7 THEN GOSUB 730
200 GOTO 40
210 REM
```

```
220 REM ----------CREATION--------------------
230 REM
240 FOR I=1 TO N
250 INPUT "NOM ";N$(i)
260 INPUT "PRENOM";p$(i)
270 INPUT "ADRESSE";A$(i)
280 INPUT "VILLE ";v$(i)
290 INPUT "TEL";T$(i)
300 INPUT "1 ON CONTINUE 2 ON ARRETE ";CH
310 IF CH=1 THEN NEXT ELSE RETURN
320 REM
330 REM -----------RECHERCHE D APRES LE NOM -----
340 REM
350 INPUT "entrez le nom de la personne"; NR$
360 INPUT "entrez le prénom";PR$
370 FOR I=1 TO N
380 IF NR$=N$(i) AND PR$=p$(i) THEN PRINT  A$(i),V$(i),T$(i)
390 NEXT
400 RETURN
410 REM
420 REM -------RECHERCHE D APRES LE TELEPHONE----
430 REM
440 INPUT "entrez le TEL ";TR$
450 FOR i=1 TO N
460 IF T$(i)=TR$ THEN PRINT n$(i),p$(i),a$(i),v$(i)
470 NEXT
480 RETURN
490 REM
500 REM -----RECHERCHE D APRES LA VILLE ----------
510 REM
520 INPUT "entrez la ville ";VR$
530 FOR I=1 TO N
540 IF V$(i)=VR$ THEN PRINT n$(i),p$(i),t$(i)
550 NEXT
560 RETURN
570 REM
580 REM --------EFFACEMENT ----------------------
590 REM
600 INPUT "entrez le nom à effacer ";NF$
610 INPUT "entrez le prénom a effacer ";PF$
620 FOR I=1 TO N
630  IF N$(i)=NF$ AND p$(i)=PF$ THEN N$(i)="":P$(i)="": A$(i)="": v$(i)=""
:t$(i)=""
640 NEXT
650 RETURN
660 REM
670 REM -------ENTREE D UN NOM ----------------
680 REM
690 FOR i=1 TO N
700  IF  N$(i)=""  THEN  INPUT  "nom";n$(i):INPUT  "prenom  ";p$(i):INPUT
"adresse";a$(i):INPUT "ville";v$(i):INPUT " tel";t$(i):GOTO 720
710 NEXT
720 RETURN
730 REM
740 REM --------LECTURE----------------------
750 REM
760 FOR i=1 TO N
770 IF N$(i)="" THEN 790
780 PRINT N$(i);TAB(20);P$(i);TAB(40);A$(i);TAB(60);v$(i);TAB(80);t$(i)
790 NEXT
800 RETURN
```

Commentaires

Rien de particulier ici. On voit simplement que pour effacer un nom, on entre ''''à la place, c'est-à-dire rien.

7.
DIVERSES PETITES CHOSES

7.1. LECTURE DU CLAVIER ET CHOIX

INKEY$

On a remarqué qu'à la lecture d'un menu correspond évidemment un choix et que ce dernier propose à l'utilisateur d'entrer soit un nombre soit une lettre comme ceci par exemple :

```
    MENU
1   LECTURE
2   RECHERCHE
3   TRI
    entrez la valeur choisie
```

Nous avons pu résoudre facilement ce problème en se servant des instructions à notre disposition ; il en est une cependant qui simplifie la saisie du choix en évitant de frapper et le numéro choisi et la touche RETURN. Cette fonction permet de lire le clavier, INKEY$ (de input key, entrer le mot-clé). On peut l'utiliser ainsi :

```
10 PRINT " MENU"
20 PRINT " 1 LECTURE
30 PRINT " 2 RECHERCHE
40 PRINT " 3 TRI
```

```
50 PRINT "entrez la valeur choisie"
60 ENTREE$=INKEY$
70 IF A$="1" THEN 1000
80 IF A$="2" THEN 2000
90 IF A$="3" THEN 3000
100 GOTO 60
200 ...
1000 REM  lecture
1030 ...
2000 REM  recherche
2030 ...
3000 REM  tri
3030 ...
```

Les lignes qui nous intéressent ont le numéro 60 et 100. En 60, la valeur ENTREE$ est une valeur alphanumérique qui lit le clavier, en d'autres termes qui attend que l'utilisateur frappe une touche. Si cette valeur est "1" ou "2" ou "3" un test déroute le pointeur vers un sous-programme situé en ligne 1000, 2000 ou 3000 ; si cette valeur est différente, la ligne 100 renvoie de nouveau à la lecture du clavier. On voit qu'il n'est plus nécessaire d'entrer "1" et RETURN, mais seulement d'entrer "1" pour aller en ligne 1000 : la saisie d'informations est donc plus aisée et plus rapide.

7.2. LA GESTION DES ERREURS

ON ERROR GOTO, RESUME NEXT, ERR, ERL

Nul n'étant parfait, on peut faire des erreurs en programmant. Le Basic renvoie alors un message approprié et l'exécution s'arrête sur la ligne erronée. On peut cependant gérer les erreurs en déroutant le pointeur vers un sous-programme créé par le programmeur avec les instructions ON ERROR GOTO, placées en tête du programme, et RESUME, ERL, ERR ; ainsi :

```
1 ON ERROR GOTO 1000
10 PRINT "L lecture
20 PRINT "R recherche
30 PRINT "T tri
40 A$= INKEY$
```

```
50 IF A$="L" THEN  90
60 IF A$="R" THEN 110
70 IF A$="T" THEN 130
80 GOTO 40
90 REM lecture
100 ...
110 REM Recherche
120 ...
130 REM tri
140 ...
999 STOP
1000 PRINT " Erreur n°";ERR; " en ligne n°";ERL
1020 PRINT " Vous avez écrit des petits points: c'est
           très vilain! "
1030 RESUME 10
```

La ligne 1 sert à prévenir qu'en cas d'erreur l'exécution ne s'arrête pas mais est déroutée vers la ligne 1000. En ligne 1000 et suivantes se trouve la gestion des erreurs. Deux variables sont réservées à cela, ERR (de error, erreur) qui renvoie le numéro de l'erreur et ERL (de error line, ligne de l'erreur), qui renvoie le numéro de la ligne où se trouve l'erreur. Le numéro de l'erreur correspond à un code relativement standard des ordinateurs et est répertorié page 353 du manuel livré avec l'AMSTRAD PCW 8256. Par exemple, si le pointeur rencontre l'instruction NEXT sans avoir au préalable rencontré l'instruction FOR, il renvoie le message "Unexpected NEXT", dont le numéro de code est 1.

Si l'on fait le programme donné en exemple, on va s'apercevoir que les lignes 100, 120 ou 140 constituées de petits points provoquent une erreur de syntaxe, de numéro de code 2.

La ligne 1030 est importante. L'instruction RESUME (de resume, reprendre) peut s'écrire de trois manières différentes :

— Avec un numéro de ligne, comme dans l'exemple, c'est équivalent à un GOTO 10 ;

— Avec l'extension NEXT, soit : RESUME NEXT ; c'est équivalent à un RETURN, c'est-à-dire de repartir à l'instruction suivant l'erreur ;

— sans extension, renvoie à l'instruction où se trouve l'erreur.

7.3. UNE AUTRE FAÇON DE CHOISIR

ON... GOTO ON... GOSUB

On a remarqué que, lors d'un choix, il faut écrire un certain nombre de tests renvoyant à des sous-programmes. Plus il y en a, plus il y de lignes de tests. On peut éviter de surcharger le choix en utilisant ON... GOTO ou ON... GOSUB. Ainsi, si on a :

```
10 PRINT "1 LECTURE
20 PRINT "2 TRI
30 PRINT "3 RECHERCHE
40 PRINT "4 AJOUT
50 PRINT "5 EFFACEMENT
60 PRINT "6 CALCULS
70 INPUT "entrez votre choix";C
80 IF C=1 THEN 1000
90 IF C=2 THEN 2000
100 IF C=3 THEN 3000
110 IF C=4 THEN 4000
120 IF C=5 THEN 5000
```

On peut écrire, en supprimant les lignes 80 à 120 :
80 ON C GOTO 1000, 2000, 3000, 4000, 5000.

La fonction ON... GOTO dispose d'un compteur interne calqué sur la valeur de C. Quand C vaut 1, le système pointe sur la première valeur après GOTO et ainsi de suite. L'instruction ON... GOSUB est rigoureusement identique mais fonctionne bien sûr comme GOSUB.

7.4. LA SEGMENTATION DES PROGRAMMES

CHAIN, CHAIN MERGE, ALL, COMMON

Il peut être utile parfois, lors de l'élaboration de gros programmes, de les segmenter en plus petits morceaux indépendants les uns des autres ; c'est une notion déjà rencontrée avec les sous-programmes. Cependant, même ainsi on peut rencontrer des difficultés notamment sur la place restant disponible en mémoire. Il convient alors de segmenter un programme en plusieurs sous-programmes et d'utiliser ces

derniers seulement quand on en a besoin. En d'autres termes, chacun de ces sous-programmes ne se trouve pas en mémoire centrale mais reste stocké sur la disquette. Ainsi, quand le besoin s'en fait sentir, et seulement à ce moment-là, un ordre est donné d'aller chercher le sous-programme en question et de la charger en mémoire vive.

Cet ordre est CHAIN (de chain, chaîne). Quand il est rencontré, il oblige le système à charger un second programme spécifié. Il y a plusieurs façons de l'utiliser. Il n'est donné ici que la principale, avec l'option MERGE.

Supposons avoir écrit et sauvegardé sous le nom "PREMIER" le programme suivant :

```
10 REM PREMIER PROGRAMME
20 PRINT "ceci est le premier programme qui va de la
          ligne 10 à 50"
30 PRINT
40 A=10
50 CHAIN MERGE "DEUX",60,ALL,DELETE 10-50
```

Puis faisons de même avec le programme suivant nommé "DEUX" :

```
60 REM SECOND PROGRAMME
70 PRINT "ceci est le second programme qui va de la
          ligne 60 à 100"
80 PRINT
90 PRINT A
100 CHAIN MERGE "PREMIER",10,ALL,DELETE 60-100
```

Rechargeons avec LOAD le premier :

LOAD "PREMIER"

et lançons-le ;

RUN

ceci est le premier programme qui va de la ligne 10 à 50

ceci est le second programme qui va de la ligne 60 à 100

10

ceci est le premier programme qui va de la ligne 10 à 50

ceci est le second programme qui va de la ligne 60 à 100

Arrêtons avec STOP l'exécution. On vient de voir l'exécution successive de deux programmes qui s'appellent l'un l'autre. Chacun s'exé-

cute puis charge le suivant et disparaît de la mémoire. Tout est contenu dans les 50 et 100. Ici, CHAIN MERGE charge le programme nommé, l'exécute à partir de la ligne spécifiée, 10 ou 60 selon le cas, conserve toutes les variables avec ALL (de all, tous) et efface de la mémoire le programme appelant avec DELETE.

On voit que de cette manière un programme énorme, ou dont l'exécution exige beaucoup de mémoire peut être segmenté ; en fait, il est bon de s'habituer à programmer ainsi, c'est-à-dire de structurer ses programmes.

COMMON

On doit préciser dans le programme les valeurs qui doivent transiter du premier programme au second. Ceci est fait avec l'instruction COMMON (de common, commun). Ainsi, si l'on désire garder les variables A, B, D$, on dira :

COMMON A,B,D$

COMMON RESET

Agit sur les valeurs précédentes. Cette instruction efface toutes les valeurs, sauf celles citées dans COMMON.

7.5. AIDE A LA MISE AU POINT

FRE TRON TROFF

On peut avoir besoin de connaître la mémoire disponible ; deux possibilités existent avec FRE (de free, libre) :

PRINT FRE(0)

et on connait la mémoire restant à la disposition du programme ;

PRINT FRE('''')

et on connait la mémoire maximun disponible. Ici, on force le Basic à compacter la mémoire, c'est-à-dire, à la différence de FRE(0), de rechercher les zones éventuellement libres en mémoire.

Les résultats sont donnés en octets.

Il arrive souvent que les programmes soient tellement complexes que l'on se perd dans le fouillis créé par les IF...THEN, GOSUB et autres GOTO, précisément quand le programme est peu ou pas structuré, défaut majeur des débutants. Il existe cependant deux instruc-

tions qui peuvent s'avérer utiles lors de la mise au point, TRON et TROF (de trace on et trace off, activation/désactivation de la trace).

Ainsi, la fonction TRON ne s'utilise pas dans un programme, mais avant de le lancer avec RUN. Ce faisant, nous voyons s'afficher sur l'écran, outre les résultats normaux auxquels on s'attend, le numéro des lignes analysées par le Basic. Par exemple, faisons le programme suivant :

```
10 PRINT "bonjour"
20 PRINT "au revoir"
30 PRINT " ce programme est positivement stupide"
40 PRINT " vous devriez en inventer un vous-même!"
```

puis entrons TRON et RUN :

```
§10§ bonjour
§20§ au revoir
§30§ ce programme est positivement stupide
§40§ vous devriez en inventer un vous-même!
```

Grâce à TRON, on sait que le pointeur est passé en ligne 10, 20, 30 et 40. Pour désactiver cette fonction on entrera simplement : TROFF.

8.
LE TRAITEMENT DE CHAINES DE CARACTÈRES

8.1. DÉCOUPAGE DE CHAÎNES DE CARACTÈRES

LEFT$ MID$ RIGHT$

Une des principales fonctions du Basic et en même temps une des plus passionnantes est la possibilité d'analyser les chaînes de caractères grâce à quelques instructions.

Pour obtenir les X premiers caractères d'une valeur alphanumérique, on prendra les X caractères de gauche avec LEFT$ (de left, gauche). Ainsi, nous avons :

A$ = ''BONJOUR'',

obtenir les trois premiers caractères c'est dire :

PRINT LEFT$(A$, 3) et nous aurons :

BON affiché à l'écran. Remarquez qu'on aurait pu aussi dire : PRINT LEFT$ (''BONJOUR'', 3).

Il suffit donc de préciser la valeur alphanumérique et le nombre de caractères désirés ; il faut bien comprendre qu'avec cette instruction nous obtenons une autre chaîne, et plus précisément une sous-chaîne de caractères qui s'analyse et s'utilise exactement comme une chaîne ''normale''. Ainsi pouvons-nous stocker les trois premiers caractères de A$ dans une valeur spécifiée :

GAUCHE$ = LEFT$(A$,3)
PRINT GAUCHE$ donne BON.

Ceci étant dit, le lecteur avisé aura certainement compris que, si on peut extraire à gauche, on doit pouvoir le faire à droite. Eh oui ! c'est possible avec l'instruction RIGHT$ (de right, droite) qui s'utilise comme LEFT$ en comptant les caractères à partir de la droite :

PRINT RIGHT$ (A$,4) donne

JOUR et évidemment on peut faire : DROITE$ = RIGHT$(A$,4).

Pour accéder aux caractères inclus dans la chaîne, une autre instruction existe et est certainement la plus importante puisque, bien utilisée, permettrait de se passer des deux précédentes ; c'est l'instruction MID$ (de middle, le milieu).

Il ne faut pas, pour commencer, prendre ce terme de milieu au pied de la lettre (si on peut dire !). Nous allons voir pourquoi.

Si nous voulons extraire la chaîne de caractères "gramme" de la chaîne "programmer", nous écrirons :

A$ = "programmer"
PRINT MID$(A$,4,6)
et nous obtenons :
gramme

Le premier nombre, 4, signifie l'endroit dans A$ où commence la sous-chaîne recherchée ; le second, contrairement à ce que beaucoup croient au début, n'est pas l'endroit où finit la sous-chaîne mais la longueur de cette sous-chaîne. De plus, on voit qu'il n'est pas nécessaire de choisir une sous-chaîne exactement au milieu mais n'importe où dans la chaîne. Si on dit :

PRINT MID$(A$,4,7) nous obtenons la chaîne "grammer" ; or c'est exactement équivalent à
PRINT RIGHT$(A$,7).

Par ailleurs, dire :

PRINT MID$ (A$,1,3) est exactement équivalent à :
PRINT LEFT$ (A$,3)

Ainsi MID$ peut avantageusement remplacer LEFT$ et RIGHT$.

De plus, l'instruction MID$ peut s'écrire autrement ; on peut éviter de préciser la longueur de la sous-chaîne extraite. Il suffit d'écrire :

PRINT MID$(A$,4) et nous aurons toute la chaîne A$, commençant au quatrième caractère et finissant au bout de A$:
donc ici, "grammer".

LEN

Le basic permet de connaître la longueur d'une chaîne avec l'instruction LEN (de length, la longueur). C'est un nombre qui est renvoyé et s'écrit ainsi :

PRINT LEN(A$)

L'ensemble ''LEN (A$)'' renvoyant un nombre, on peut donc écrire :

LONG = LEN(A$) par exemple.

Voyons quelques applications de ces quatre instructions.

— Première application

Nous voudrions obtenir ceci :

P
PA
PAT
PATR
PATRI
PATRIC
PATRICE

Comment analyser ce problème ? Regardons le résultat final : c'est la chaîne de caractères ''PATRICE''. Regardons maintenant chaque sous-chaîne, comptons et mesurons les ; nous obtenons :

Nom	lieu de départ	longueur
P	1	1
PA	1	2
PAT	1	3
PATR	1	4
PATRI	1	5
PATRIC	1	6
PATRICE	1	7

Conclusion : il faut décomposer une chaîne de valeur ''PATRICE'' en morceaux de longueur croissante. La répétition nous fait irrésistiblement penser à l'usage d'une boucle, et le découpage des mots aux instructions MID$ ou LEFT$; on peut donc écrire le programme de deux manières :

```
10 A$ = "PATRICE"
20 FOR I= 1 TO LEN(A$)
30 PRINT MID$(A$,1,I)
40 NEXT
```

ou bien :

30 PRINT LEFT$(A$,I)

— Seconde application

Nous désirons obtenir ceci :

D
AD
RAD
TRAD
STRAD
MSTRAD
AMSTRAD

Même analyse que précédemment, nous obtenons :

Nom	lieu de dép. (en partant de gauche)	longueur
D	7	1
AD	6	2
RAD	5	3
TRAD	4	4
STRAD	3	5
MSTRAD	2	6
AMSTRAD	1	7

Nous avons ici une valeur qui décroit et une autre qui croît de manière inverse ; on peut donc relier ces valeurs entre elles par un dénominateur commun en utilisant une boucle; cette dernière en allant de 1 à 7 pourrait symboliser la longueur croissante. Quant au lieu de départ, on peut le calculer à l'aide de la fonction suivante :

Lieu de départ = LEN (A$) + 1 − I

```
10 A$= "AMSTRAD"
20 FOR I= 1 TO LEN(A$)
30 PRINT MID$(A$,LEN(A$)+1-I,I)
40 NEXT
```

Ou bien plus simplement en comptant les caractères à partir de la fin :

```
30 PRINT RIGHT$(A$,I)
```

Cet exemple, particulièrement tordu (on l'avoue) montre cependant qu'il est possible de dépasser le cadre strict des instructions Basic.

L'opérateur " + " et la concaténation de chaînes de caractères.

Si on peut découper des chaînes, il est prévu de pouvoir les recoller. Il suffit pour cela d'utiliser le signe " + ". Soit les chaînes suivantes :

A$ = ''Salut''
B$ = ''les ptits gars !''
C$ = ''Comment ça va ?''

Pour faire une phrase de ces trois expressions éblouissantes, on fera : P$ = A$ + B$ + C$
et, en faisant PRINT P$, nous obtiendrons ce texte impérissable :

Salut les ptits gars ! comment ça va ?

Ceci se nomme une ''concaténation'' de chaînes.

UPPER$ LOWER$

Voici de simples instructions (pour nous changer les idées) qui peuvent cependant être fort utiles.

Il faut savoir que, en Basic, les minuscules sont complètement différentes des majuscules et nous verrons précisément pourquoi bientôt ; ainsi, le mot ''RECTANGLE'' et le mot ''rectangle, de même que ''RECtangle'' (sans parler de ''ReCtanGLE'' ou même de ''REctangLE'' !!!!) sont trois chaînes différentes en Basic.

Or, tout ceci est réellement important lors de comparaisons de chaînes. En effet, on peut entrer un mot en minuscules (sans le faire volontairement) et que ce dernier soit comparé au même mot écrit en majuscules : la comparaison échouera alors que nous sommes persuadés avoir fait un bon programme. Pour éviter les crises de nerf, les dépressions et autres drames divers, il existe deux fonctions réécrivant une chaîne.

Ainsi, si A$ est égale aux différentes versions du mot ''rectangle'' décrites, il suffira de dire :

A$ = UPPER$ (A$) pour transformer A$ en ''RECTANGLE''.
De la même façon, on peut transformer A$ en ''rectangle'' avec :

A$ = LOWER$ (A$) (de up, haut et low, bas).

INSTR

On peut rechercher dans une chaîne de caractères une sous-chaîne avec l'instruction INSTR (de in string, dans la chaîne) qui s'utilise de deux façons différentes :

```
A$ = ''BONJOUR LES AMIS COMMENT CA VA''
B$ = ''A''
PRINT INSTR(A$,B$)
PRINT INSTR (28,A$,B$)
```

Dans le premier cas, le système renvoie 13, car ''A'' est bien le treizième caractère de A$; dans le second, il renvoie 30, car ce nombre est bien l'emplacement du premier ''A'' dans A$, à partir du 28e caractère.

8.2. CONVERSIONS

VAL STR$

Soit les chaînes suivantes :

A$ = ''12''
B$ = ''34''

Si l'on entre :
PRINT A$
nous aurons sur l'écran :
12
et si l'on entre :
PRINT B$
nous aurons :
34

mais si on entre :
PRINT A$ + B$

que se passe-t-il ? Nous obtenons simplement la chaîne 1234 ; en effet, nous venons de faire une concaténation de deux chaînes. Et si nous avions voulu la SOMME de 12 et de 34 ? C'est impossible, sauf si l'on convertit les chaînes en nombres et cette conversion peut se faire avec l'instruction VAL (de value, valeur). On écrit alors :

PRINT VAL (A$) et cela donne :
12

Il n'y a pas apparemment de différences entre ''12'' écrit avec PRINT A$ et ''12'' écrit avec PRINT VAL(A$). En fait, il y en a une, et de taille. La première fois, nous voyons sur l'écran la chaîne de caractères ''12'' composée du caractère ''1'' et du caractère ''2'' ; la seconde fois, nous voyons le nombre 12, composé comme chacun sait des chiffres 1 et 2.

Avec VAL, nous pouvons donc faire des opérations avec des chaînes :

Ainsi :

PRINT VAL (A$) + VAL (B$) donnera bien 46. Ou bien :

C = VAL(A$) + VAL(B$)
PRINT C est également possible.

Question-piège : que vaut :
PRINT VAL(A$ + B$) ?
Simplement le nombre 1234.

Il faut noter que VAL ne peut convertir évidemment que des caractères symbolisant les chiffres. Si on entre PRINT VAL(''BONJOUR''), la réponse sera zéro.

Puisque l'on peut convertir une chaîne en nombre, le contraire est également possible avec l'instruction STR$ (de string, la chaîne) qu'il ne faut pas confondre avec STRING$:

```
B = 12
A$ = ''j'ai '' + STR$(B) + '' ans''
```

Cela donne :

```
A$ = ''j'ai 12 ans''
```

8.3 CODES

ASC CHR$

Il existe quelques normes, en informatique, nécessaires pour l'échange d'informations d'un matériel à un autre. Ces normes stipulent que les caractères utilisés doivent correspondre à un même code. Il y a donc associé à chaque caractère de l'AMSTRAD un numéro de code qui est théoriquement le même sur un autre ordinateur ou un périphérique. Nous avons accès à ce code avec la fonction ASC (de A.S.C.I.I, prononcez ''aski'', American Standard Code Information Interchange). Pour s'en servir, on entre :

PRINT ASC(''A''), par exemple, et nous obtenons :

65. En effet, ce nombre est le code de la lettre A majuscule ; on devine que B porte le numéro 66, C le 67 etc. On peut en déduire qu'une fonction inverse doit exister qui donne, à partir d'un nombre, le caractère correspondant ; nous avons cette fonction avec CHR$ (de character, caractère). Ainsi :

PRINT CHR$(65) donne
A.

La fonction CHR$ n'inspire pas le débutant en général. C'est pourtant une fonction à connaître ; par exemple :

PRINT CHR$(7) déclenche un bip sonore.

PRINT CHR$(27) + ''E'' efface l'écran. Mille fois hélas. l'instruction pourtant si courante ailleurs, CLS, n'existe pas sur l'AMSTRAD.

Explorez ce qu'il est convenu d'appeler les codes de contrôle situés de 0 à 31 en entrant PRINT CHR$(valeur) et voyez !

LPRINT CHR$(27) + CHR$(15) : écrit sur l'imprimante en petits caractères.

EXERCICES

Ex n° 1

On imagine avoir le fichier bibliothèque composé de titres d'ouvrages et d'auteurs suivant :

Madame Bovary	FLAUBERT
Tu seras tout seul dans ton cercueil	CHASE
Tintin au Tibet	HERGE
La débâcle	ZOLA
L'île au trésor	STEVENSON
L'île mystérieuse	HERGE
La révolution Française	MICHELET
La grande révolution	KROPOTKINE
L'île noire	HERGE

Le programme doit permettre l'analyse des titres du fichier à partir d'un mot-clé. Par exemple, on désire connaître tous les titres dans lesquels se trouve la chaîne de caractères "île", ainsi que l'auteur correspondant.

Ex n° 2

Le programme doit analyser une chaîne de caractères quelconques et placer dans l'ordre décroissant les caractères lettre par lettre. Par exemple, si la phrase est "un chasseur sachant chasser", nous aurons le classement suivant :

s 5, a 4, c 3, h 3 etc ; parce que "s" est la lettre la plus courante, puis "a" etc...

Ex n° 3

Le programme devra permettre la traduction d'une phrase simple à partir d'un dictionnaire composé des mots suivants :
Informatique data processing, soleil sun, le the, la the, ciel sky, dans in, sur on, maison house, brille shines, de of, pluie rain, tombe falls, voiture car, est is, route road, banche white, bleu blue, facile easy, et and (il va sans dire qu'aucune analyse linguistique n'est prévue, seulement une traduction mot à mot). On entrera un texte simple, par exemple "la voiture est sur la route de la maison blanche et le ciel est bleu" et on obtiendra "the car is on the road of the house white and the sky is blue". De plus, le système devra préciser un mot inconnu de lui.

Ex n° 4

Le programme permettra de transformer un mot écrit en majuscules en minuscules et vice-versa, sans utiliser UPPER$ et LOWER$, bien sûr !

SOLUTIONS COMMENTÉES

Ex n° 1

1 — Pour analyser un fichier, il faut que ce fichier existe ; nous allons stocker en DATA les 9 titres et les auteurs puis les entrer en un tableau TITRE$(9) et un tableau AUTEUR$(9) ;

2 — Nous entrerons le mot-clé dans MC$;

3 — Il faut analyser les titres un par un ; nous le ferons avec une boucle J ;

4 — Il faut analyser chaque titre caractère par caractère : nous le ferons avec la boucle I ;

5 — Il faut chercher dans le titre une sous-chaîne, qui est MC$, et cette chaîne commence en I et a pour longueur obligatoirement celle du mot-clé. On fera un test qui, positif, affichera l'auteur.

```
10 DIM  TITRE$(9),AUTEUR$(9)
20 DATA madame bovary,flaubert,tu seras tout seul dans ton
cercueil,chase,tintin au tibet,hergé,la débacle,zola,l'ile au
trésor,stevenson,l'ile mystérieuse,hergé,la révolution
francaise,michelet,la grande révolution,kropotkine,l'ile noire,hergé
30 FOR I=1 TO 9
40 READ TITRE$(i),AUTEUR$(i)
50 NEXT
60 INPUT "entrez le mot-clé ";MC$
70 FOR j=1 TO 9
80  FOR I=1 TO LEN(TITRE$(J))
90  IF MID$(TITRE$(J),I,LEN(MC$))= MC$ THEN PRINT AUTEUR$(J),TITRE$(J)
100 NEXT
110 NEXT
```

Ex n° 2

```
10 REM
20 DIM A(26),let$(26)
30 INPUT "entrez votre texte ";A$
40 A$=UPPER$(A$)
50 FOR i=1 TO 26
60 LET$(I)=CHR$(64+I)
70 NEXT
80 FOR J=1 TO  26
90 FOR I= 1 TO LEN(A$)
100 IF MID$(A$,I,1)=LET$(J) THEN A(J)=A(J)+1
110 NEXT
120 NEXT
```

```
130 REM
140 REM TRI
150 REM
160 FOR J=1 TO 25
170 FOR I=1+J  TO 26
180  IF A(I) <= A(J) THEN 200
190   SWAP A(i),A(j):SWAP LET$(i),LET$(J)
200 NEXT
210 NEXT
220 REM AFFICHAGE
230 LPRINT A$
240 FOR I=1 TO 26
250 IF A(i)= 0 THEN 270
260 LPRINT LET$(i), A(i)
270 NEXT
```

Commentaires

On emplit un tableau LET$ des lettres de l'alphabet grâce à CHR$; puis on trie : notez le tri nommé "tri par sélection".

Ex n° 3

```
10 REM TRADUCTION AUTOMATIQUE
20 DIM F$(19),A$(19)
30 DATA informatique,data processing,soleil, sun,le,the ,la,the,ciel
,sky,dans, in,sur,on,maison,house,brille,shines,de,of,pluie ,rain,tombe
,falls,voiture,car,est,is,route,road,blanche,white,l',the, facile, easy
,et,and
40 FOR I=1 TO 19
50 READ F$(i),A$(i)
60 NEXT
70 INPUT "entrez la phrase ";P$
80 p$=LOWER$(p$)
90 LPRINT P$
100 REM TRADUCTION
110 P$=P$+" "
120 DEBUT=1
130 FOR I=1 TO LEN(P$)
140 IF MID$(P$,i,1)= " " THEN N$= MID$( P$, DEBUT,( I-DEBUT)):GOSUB
180:DEBUT=I+1
150 NEXT I
160 STOP
170 REM DICTIONNAIRE
180 FOR J=1 TO 19
190 IF N$=F$(J) THEN LPRINT A$(J);" ";:GOTO 220
200 NEXT J
210 LPRINT "XXX";" ";
220 RETURN
```

Commentaires

On créé un dictionnaire, on entre la phrase à traduire et on cherche chaque mot de cette phrase pour le comparer au dictionnaire. Cette recherche se fait en positionnant un pointeur nommé DEBUT qui

symbolise la première lettre de chaque mot et la boucle I symbolisera la fin du mot. En ligne 140, on recherche donc chaque mot : I sera à la fois la fin d'un mot et le début du suivant, d'où DEBUT = I + 1.

Si le mot à traduire n'existe pas, on écrira à la place "XXX".

Ex n° 4

```
10 INPUT "entrez un mot ";P$
20 FOR I=1 TO LEN(P$)
30 IF ASC( MID$(P$,i,1)) > 96 THEN PRINT CHR$(  ASC( MID$(P$,i,1))-32);
40 IF ASC( MID$(P$,i,1)) < 96 THEN PRINT CHR$(  ASC( MID$(P$,i,1))+32);
50 NEXT
```

Commentaires

Relativement simple. On se sert de la différence des codes ASCII entre les majuscules et les minuscules.

9.
LES FICHIERS SUR DISQUE

INTRODUCTION

Nous avons vu jusqu'à présent comment manipuler des données ; nous avons appris à stocker des informations dans un programme avec DATA et de manière générale avec la notion de valeur constante. De plus, nous avons appris à entrer les valeurs en tableau pour permettre une analyse plus facile, pour permettre des calculs ; et puis, pour pouvoir réutiliser les programmes, nous avons appris à les sauver et à les recharger avec SAVE et LOAD. Cependant, tout ce que nous avons vu dans cet ouvrage ne présente quasiment aucun intérêt si nous ne pouvons pas réutiliser toutes ces données, toutes ces informations, entrées au clavier le long de cet ouvrage, ou d'autres. Il nous faut donc maintenant apprendre comment garder les informations pour pouvoir les utiliser ultérieurement.

Pour conserver des données, on utilise depuis longtemps ce qu'il est convenu d'appeler les fichiers. On se sert d'un classeur ou d'un meuble spécialement conçu à cet effet pour pouvoir engranger, classer, sélectionner des renseignements divers. Les avantages en sont connus, les inconvénients aussi : place de plus en plus grande au fur et à mesure de l'arrivée de nouveaux renseignements, difficulté accrue d'analyse et de rangement.

Avec l'ordinateur, tous ces problèmes disparaissent rapidement, car on peut stocker sur un disque toutes les données voulues. Il va sans dire que le nombre d'informations mémorisées dépend de la capacité de la disquette. Sur l'AMSTRAD PCW 8256, la disquette peut contenir environ 170 kilo-octets, soit 170.000 caractères, par face. Le second lecteur optionnel permet 720 Kilo-octets accessibles en même temps et peut-être dans l'avenir un disque dur de 10 ou 20 Mega-octets (10 à 20 millions d'octets) sera disponible. Quand on sait qu'une let-

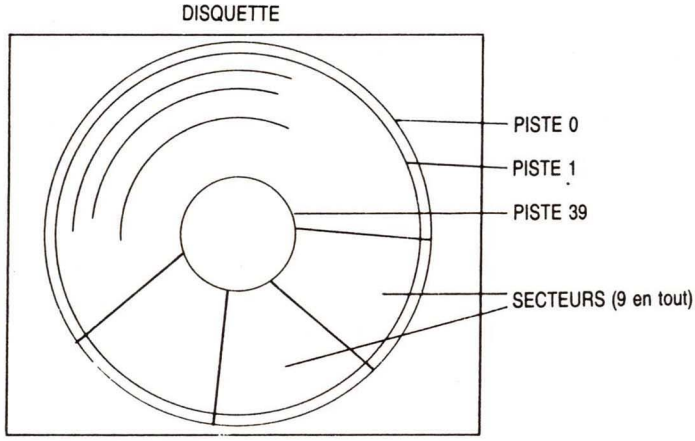
tre de 90 colonnes sur 30 lignes représente 90* 30 = 2700 caractères, on voit qu'un ouvrage comme celui-ci tiendrait sur une disquette (à noter qu'il tient réellement sur une disquette, puisque l'auteur a utilisé le traitement de texte LOCOSCRIPT).

La disquette

Une disquette est un ensemble magnétique destinée à la conservation des données. Après avoir été formatée (cf. chapitre 3), elle a 40 pistes, qui sont des cercles concentriques, numérotées de 0 à 39 et est divisée en 9 secteurs identiques ; les informations essentielles concernant les noms et coordonnées des fichiers et programmes sont stockées sur les pistes 0 et 1 dans un endroit nommé DIRECTORY, le catalogue. Nous pouvons lire ces informations en faisant DIR ou FILES.

Les disquettes sont repérées par leur taille, donnée en anglais ; celle qui nous concerne mesure 3 pouces. Par ailleurs, on peut les protéger d'un effacement intempestif en enfonçant la languette de protection. Le lecteur est quant à lui muni d'une tête de lecture-écriture qui inscrit ou lit les informations de manière magnétique, un peu comme les bandes magnétiques audio.

DISQUETTE



Il faut savoir également que l'AMSTRAD PCW 8256 possède un second lecteur situé en mémoire vive, appelé M qui s'utilise exactement comme le lecteur A.

Qu'est-ce-qu'un fichier en informatique ?

Tout d'abord, une précision s'impose : le fichier en anglais se dit "file" et de façon générale, c'est ainsi qu'on appelle un fichier au sens strict en français mais également un programme ; le terme "file" est donc employé indifféremment pour signifier fichier et programme.

Un fichier est une collection d'informations de même nature, destinée à stocker des informations, dans le but de les relire ultérieurement et éventuellement de les modifier. En informatique, on distingue deux types principaux de fichiers selon la méthode utilisée pour les stocker et les relire : il y aura donc ce qu'il est convenu d'appeler "fichier à accès séquentiel" et "fichier à accès direct" qui formeront la moitié environ de ce chapitre. Mais il existe également une autre manière d'utiliser plus rationnellement et plus efficacement le Basic pour manipuler des informations, qui s'appelle "fichier à accès indexé" ; nous pouvons en Basic créer avec les instructions apprises dans cet ouvrage ce que l'on appelle un index et nous apprendrons à le faire. Mais il faut savoir que le Basic de l'AMSTRAD PCW 8256 possède un gestionnaire intégré de fichier à accès indexé nommé JET-SAM qui permet d'optimiser la gestion de fichiers. Ceci est une méthode relativement nouvelle car peu connue, même sur des ordinateurs plus puissants. Nous apprendrons donc une trentaine de mots Basic nouveaux et propres à l'AMSTRAD.

9.1. LES FICHIERS A ACCÈS SÉQUENTIEL

Un fichier est un ensemble de fiches comportant un certain nombre d'informations. Par exemple, ce livre est un fichier composé de chapitres classés les uns derrière les autres. Supposons que nous voulons lire tout ce qui a trait aux fichiers et que, malheureusement, l'éditeur du livre a oublié (!) d'inclure le sommaire ; que devons-nous faire pour trouver le chapitre en question ? Une seule solution : lire chapitre par chapitre jusqu'à trouver le bon. Nous venons de pratiquer un accès séquentiel, c'est-à-dire que pour trouver une information précise, et à supposer qu'elle existe, il a fallu commencer par la première information et lire séquence par séquence jusqu'au bon résultat. Ceci est une première caractéristique des fichiers à accès séquentiel : c'est dire qu'ils portent bien leur nom !

9.1.1. La création

OPEN "O", PRINT #, WRITE # CLOSE FIND$ RESET

Prenons un exemple simple et proposons-nous la création d'un fichier téléphonique, comportant les rubriques suivantes :

Le nom, l'adresse, le numéro de téléphone (à noter : En informatique, nous appellerons une rubrique un champ et un ensemble de champs un enregistrement : un fichier est donc un ensemble d'enregistrements).

Pour créer le fichier nous devons donner l'ordre à l'ordinateur avec l'instruction OPEN (de open, ouvrir) d'ouvrir le fichier. En accès séquentiel, il faut préciser si cette ouverture est destinée à créer ou lire le fichier avec la lettre "i"ou lettre "o" (de Input ou Output, entrée ou sortie). Puis il faut préciser le numéro de fichier utilisé avec le signe "#". Il faut savoir qu'une information à écrire sur disque ne peut se faire qu'en réservant une mémoire-tampon, dite "buffer", par laquelle transitent les informations. Cette mémoire est créée avec la commande OPEN et n'a pas grande influence dans la programmation mais nous verrons avec les autres types de fichier qu'il faudra en tenir compte. Pour l'instant, il faut retenir qu'il est nécessaire de numéroter un buffer par fichier ouvert : si deux fichiers sont utilisés simultanément, on ouvrira donc un buffer pour chacun de ces fichiers.

Pour créer ou lire un fichier, encore faut-il savoir de quel fichier il s'agit et sur quel lecteur il se trouve : nous le préciserons donc. Tout ceci donne, pour créer le fichier téléphone nommé "TEL" sur le lecteur A :

```
100 OPEN "O", #1, "A:TEL"
```

Nous reconnaissons le "O" pour créer, le numéro de fichier, le nom du lecteur séparé par le nom du fichier TEL ; on peut noter que le lecteur A étant le lecteur par défaut, c'est-à-dire le lecteur qui est lu en priorité par le système, il n'est pas nécessaire de le préciser et on aurait pu écrire la ligne 100 ainsi :
100 OPEN "O", #1, "TEL" ou bien

100 OPEN "O", #2, "M :TEL" si on veut numéroter le buffer en 2 et user du disque virtuel M (on n'oubliera pas que s'il survient une coupure de courant ou bien si tout bonnement on éteint l'ordinateur, les informations contenues en M s'envoleront immédiatement !).

Une précision s'impose ici concernant le signe #. En effet, s'il est obligatoire dans les autres versions du Basic, il ne l'est pas ici. Nous le mettrons cependant systématiquement pour ne pas dérouter le lecteur utilisant d'autres d'ordinateurs.

Pour écrire des informations sur disque, il faut d'abord les entrer en mémoire avec INPUT :

```
110 INPUT "entrez le nom ";N$
120 INPUT "entrez l'adresse ";A$
130 INPUT "entrez le téléphone";T$
```

Pour écrire maintenant ces informations sur disque, on utilise simplement PRINT #1, ou PRINT #2 selon le numéro de fichier :

140 PRINT #1, N$;'';'';A$;'';'';T$

On remarquera l'écriture peu élégante de cette ligne, ce qui nous amène à considérer un aspect important de l'accès séquentiel. En effet, il faut savoir que les informations en question vont s'écrire sur la disquette séquentiellement (on le savait déjà), sans considération de la longueur des enregistrements ; par exemple, si on enregistre plusieurs personnes, nous aurons sur la disquette :

```
"DUPONT","13 rue de la paix","23.23.12.33","DURAND","28 Avenue du
Messie","10.11.22.33","DE LA TOUR D'AUVERGNE ET DES AURES REUNIS","1 rue
DANLEBRANCAR","44.22.11.66" etc.
```

Cet exemple montre en même temps l'avantage et l'inconvénient de l'accès séquentiel. Avantage car les champs, les rubriques composant un enregistrement peuvent être de longueur indéterminée, et variante ; inconvénient car cette qualité est en même temps un défaut : ne sachant pas la longueur de chaque enregistrement, par définition fluctuante, le système ne peut trouver un enregistrement précisé par son numéro. En d'autres termes, si on cherche le 234ème enregistrement, il faut impérativement lire, donc chercher sur le disque, les 233 précédents, de manière séquentielle.

Nous disions que l'écriture de la ligne 140 était peu élégante, et de surcroît fastidieuse. On peut simplifier avec l'instruction WRITE (de write, écrire) :

```
140 WRITE #1,N$,A$,T$
```

Pour continuer à entrer d'autres données ou bien pour arrêter, on peut faire un test :

```
150 INPUT " 1 on continue 2 on arrête " ; C: IF C=1 THEN 110
```

Si on arrête, il faut refermer le fichier avec l'instruction CLOSE (de close, fermer) :

160 CLOSE #1

On notera que l'on peut fermer avec CLOSE sans numéro de buffer spécifié : le système fermera automatiquement TOUS les fichiers éventuellement ouverts.

Relisons donc le programme de création :

```
100 OPEN "O",#1,"TEL"
110    INPUT "entrez le nom";N$
120    INPUT "entrez l'adresse";A$
130    INPUT "entrez le téléphone";T$
140   WRITE #1,N$,A$,T$
150   INPUT "1 on continue 2 on arrête ";C: if C=1 THEN 110
160 CLOSE #1
```

Il ne faut évidemment pas oublier de mettre une disquette formatée dans le lecteur ! Le lecteur perspicace a remarqué qu'on utilise toujours les mêmes variables N$, A$ et T$ pour des données différentes. Ceci est logique car, dès que le contenu des trois variables est écrit sur le disque, on peut sans dommage aucun emplir de nouveau ces variables d'un autre contenu.

Une instruction intéressante à utiliser et peu courante dans d'autres versions du Basic est FIND$ (de find, trouver) ; elle sert à vérifier si un fichier ou un programme existe déjà sur le disque, sans pour autant faire FILES ou DIR, et sans interrompre le programme. On peut l'utiliser ainsi, en supposant que l'on cherche le fichier "FIC" :

IF FIND$("FIC") <>" " then PRINT "FIC existe déjà"

Une autre instruction particulièrement intéressante est la fonction RESET (réinitialiser) ; elle est importante si on a changé de disque pour signaler à l'ordinateur le changement (ne pas confondre avec RSET que l'on verra par la suite).

9.1.2. La lecture

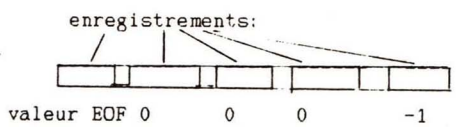
OPEN "I", EOF, INPUT #, LOF

Pour relire (séquentiellement) toutes les informations stockées, il faut de nouveau ouvrir le fichier, exactement comme en ligne 100, mais en spécifiant maintenant la lecture, donc avec "I" :

```
200 OPEN "I", #1,"TEL"
```

Pour relire les informations, il suffit de faire une boucle et de sortir un par un les enregistrements avec l'instruction appropriée, INPUT #. Or, pour faire une boucle utilisant FOR... NEXT, il faut connaître

le nombre d'enregistrements stockés sur disque : ce n'est pas souvent le cas. On va donc utiliser pour cela les instructions WHILE et WEND. Cependant, il faut quand même pouvoir déterminer la fin d'un fichier, sinon le message d'erreur ''INPUT PAST END'' (lecture du fichier après la fin) apparaît. Il existe une variable réservée nommée EOF (pour End Of File, fin de fichier) qui ne peut prendre que deux valeurs logiques, VRAI ou FAUX : – 1 si on lit un fichier et qu'on est arrivé à la fin de ce fichier, et 0 dans l'autre cas. Ainsi :




```
210 WHILE NOT EOF(1)
220  INPUT #1, N$,A$,T$
230  PRINT "nom ";N$,"adresse ";A$,"Tel ";T$
240 WEND
250 CLOSE #1
```

On voit que la ligne 210 spécifie : tant que EOF (de numéro de fichier 1) est différent de – 1 (valeur vraie), alors chercher sur disque (INPUT #1) les informations et les afficher.

On peut connaître la longueur d'un fichier avec LOF (de length of file, longueur de fichier), si celui-ci est inférieur à 128 enregistrements de 128 octets :

PRINT LOF (n° de fichier)

Cette instruction est peu intéressante en réalité et nous apprendrons à compter et conserver le nombre d'enregistrements autrement particulièrement en accès direct.

9.1.3. Les modifications

KILL NAME AS

Un problème gênant apparait dès que l'on veut modifier un fichier à accès séquentiel, soit pour effacer, soit pour ajouter un enregistrement.

En effet, le fichier étant constitué d'enregistrements de longueur indéterminée, on ne peut effacer un enregistrement de 30 caractères par exemple et le remplacer par un enregistrement de 50 : le nouvel enregistrement ne peut s'insérer à la place qui serait laissée vacante

par le précédent. Il n'existe qu'une solution : c'est créer un fichier transitoire.

Si on désire entrer un nouvel enregistrement, il faut recopier le fichier existant dans un autre, créé pour la circonstance, puis ajouter au second fichier le nouvel enregistrement. Du coup, nous aurons deux fichiers : l'ancien et le nouveau. Nous effacerons l'ancien et nous garderons le nouveau ; comme le nouveau a un autre nom, et que nous voulons garder le nom de l'ancien, nous rebaptiserons le nouveau du nom de l'ancien. C'est clair ? (Relisez ce passage !). Nous pouvons procéder ainsi :

```
300 OPEN "I", #1,"TEL"
310 OPEN "O", #2,"TEL2"
320 WHILE NOT EOF(1)
330    INPUT #1,N$,A$,T$
340    WRITE #2,N$,A$,T$
350 WEND
360    INPUT "entrez le nom";N$
370    INPUT "entrez l'adresse";A$
380    INPUT "entrez le téléphone";T$
390   WRITE #2,N$,A$,T$
400 INPUT "1 on continue 2 on arrête ";C:IF C=1 THEN 360
410 CLOSE
420 KILL "TEL
430 NAME "TEL2" AS "TEL"
```

En lignes 300 et 310, on ouvre le fichier "TEL" en lecture et le fichier "TEL2", qui n'existe pas, en création. Les lignes 320 à 350 servent à recopier "TEL" dans "TEL2". Les lignes 360 à 400 sont déjà connues. En 410, on ferme les deux fichiers (on aurait pu écrire CLOSE #1, #2, ou CLOSE #1 : CLOSE #2). La ligne 420 efface le fichier "TEL" avec KILL (de kill, tuer !) et renomme le fichier "TEL2" du nom du fichier "TEL" avec NAME... AS (nomme...comme...).

On remarquera que, outre l'inconvénient du temps perdu à attendre que la recopie soit faite, et qui peut être longue, il faut que la disquette ait la place nécessaire : si le fichier "TEL" fait 100 K-octets, il tient sur une disquette de 170 K-octets, mais si on le recopie, 100 + 100 = 200 K-octets : c'est incopiable !

Pour effacer une information on use du même procédé. Il faut recopier toutes les informations de "TEL" dans "TEL2", SAUF l'enregistrement ou les enregistrements à effacer ; il faut donc que le système connaisse le nom de la personne à effacer au préalable :

```
500 INPUT "entrez le nom de la personne à ôter du fichier ";NP$
510 OPEN "I",#1,"TEL"
520 OPEN "O",#2,"TEL2"
530    WHILE NOT EOF(1)
```

```
540       INPUT #1,N$,A$,T$
550       IF NP$ <> N$  THEN WRITE #2,N$,A$,T$
560     WEND
570 CLOSE
580 KILL "TEL"
590 NAME "TEL2" AS "TEL"
```

Il n'y a rien de particulier à signaler : un test permet de passer outre l'enregistrement à effacer : attention : ici, le programme efface toutes les personnes portant le même nom !

Exercices

Ex n° 1

Faire le programme gérant un lycée. Il permettra de :

— lire tout le fichier
— rechercher un élève par son nom, son prénom, son adresse ou sa classe
— supprimer un élève du fichier
— ajouter un élève au fichier
— de lire chaque classe triée par ordre alphabétique

On entrera dans le fichier le nom, le prénom, l'adresse et la classe de chaque élève.

Ex n° 2

Faire le programme gérant un commerce. Il permettra de :

— lire un fichier stock
— lire un fichier fournisseurs
— passer une commande par courrier
— entrer le produit à vendre

On créera un fichier stock dans lequel chaque produit sera référencé par sa désignation, son fournisseur, la quantité initiale du produit en stock, la côte d'alerte provoquant la commande au fournisseur, le prix unitaire. Le programme doit être conçu comme une caisse enregistreuse : le commercant, à chaque fois qu'il vend un produit, entrera sa désignation et la quantité vendue : le programme alors décrémentera d'autant le fichier stock au produit en question. Puis on devra pouvoir passer des commandes automatiques : le système, connaissant le stock initial et le stock actuel provoquera la commande automatique aux fournisseurs.

Solutions

Ex n° 1

```
 10 REM ----------------------LYCEE-----------------------------------
 20 REM
 30 PRINT "1 ENTREE D UN ELEVE
 40 PRINT "2 EFFACEMENT D UN ELEVE
 50 PRINT "3 LECTURE SEQUENTIELLE DU FICHIER
 60 PRINT "4 LECTURE PAR CLASSE TRIEE
 70 PRINT "5 RECHERCHES DIVERSES
 80 PRINT "6 CREATION DU FICHIER
 90 INPUT "entrez votre choix ";C
100 ON C GOSUB 130,270,430,530,830,1080
110 GOTO 30
120 REM
130 REM -------------------AJOUT  D UN ELEVE----------------------------
140 REM
150 OPEN "o",#1,"LYCEE2
160 OPEN "i", #2,"LYCEE
170 WHILE NOT EOF(1)
180 INPUT #2,n$,p$,a$,c$
190 WRITE #1,n$,p$,a$,c$
200 WEND
210 GOSUB 970
220 CLOSE
230 KILL "LYCEE
240 NAME "LYCEE2" AS " LYCEE
250 RETURN
260 REM
270 REM -----------------EFFACEMENT D UN ELEVE ------------------------
280 REM
290 INPUT "entrez le nom de l 'el)ve ";NE$
300 INPUT "entrez le prénom de l'él)ve";PE$
310 OPEN "o",#1,"LYCEE2
320 OPEN "i", #2,"LYCEE
330 WHILE NOT EOF(1)
340 INPUT #2,n$,p$,a$,c$
350 IF N$=NE$ AND P$=PE$ THEN 370
360  WRITE #1,n$,p$,a$,c$
370 WEND
380 CLOSE
390 KILL "LYCEE
400 NAME "LYCEE2" AS "LYCEE
410 RETURN
420 REM
430 REM ----------------LECTURE SEQUENTIELLE DU FICHIER ---------------
440 REM
450 OPEN "i",#1,"LYCEE
460 WHILE NOT EOF(1)
470 INPUT #1,n$,p$,a$,c$
480 PRINT n$ ;TAB(10);p$;TAB(30);A$;TAB(50)c$
490 WEND
500 CLOSE
510 RETURN
520 REM
530 REM --------------------CLASSES TRIEES-------------------------------
540 REM
550 N=100
560 V=1
570 DIM NOM$(N),PRE$(N),AD$(N),CL$(N)
580 OPEN "i",#1,"LYCEE
590 WHILE NOT EOF(1)
600 INPUT #1,n$,p$,a$,c$
610 NOM$(V)=N$
620 PRE$(V)=P$
630 AD$(V)=A$
```

```
640 CL$(V)=C$
650 PRINT NOM$(v),pre$(v),ad$(v),cl$(v)
660 V=V+1
670 WEND
680 CLOSE
690 FOR J=1 TO V-1
700 FOR I=1 TO V-2
710 IF NOM$(I) > NOM$(I+1) THEN SWAP NOM$(I),NOM$(i+1):SWAP
PRE$(I),PRE$(I+1):SWAP AD$(I),AD$(I+1):SWAP CL$(I),CL$(I+1)
720 NEXT
730 NEXT
740 INPUT "entrez la classe désirée ";CD$
750 FOR I=1 TO V-1
760 IF CD$=CL$(i) THEN PRINT NOM$(i) ;TAB(10);pre$(i);TAB(30);Ad$(i)
770 NEXT
780 INPUT "autre classe triée ?? RETURN/OUI  autre touche/NON ";Ch$
790 IF CH$="" THEN 740
800 ERASE NOM$,pre$,ad$,cl$
810 RETURN
820 REM
830 REM ------------------RECHERCHES DIVERSES ---------------------
840 REM
850 INPUT "RECHERCHES SUR: 1/NOM  2/PRENOM 3/ADRESSE 4/CLASSE";R
860 INPUT "entrez le terme recherché ";TR$
870 OPEN "i", #1,"LYCEE
880 WHILE NOT EOF(1)
890  INPUT #1,N$,P$,A$,C$
900  IF R=1 AND N$=TR$ THEN PRINT N$,P$,C$
910  IF R=2 AND P$=TR$ THEN PRINT N$,P$,C$
920  IF R=3 AND A$=TR$ THEN PRINT N$,P$,C$
930  IF R=4 AND C$=TR$ THEN PRINT N$,P$,C$
940 WEND
950 CLOSE
960 RETURN
970 REM --------------------ENTREE D UN ELEVE --------------------------
980 REM
990 INPUT "entrez le nom.............";N$
1000 INPUT "entrez le prénom..........";P$
1010 INPUT "entrez l'adresse..........";A$
1020 INPUT "entrez la classe..........";C$
1030 WRITE #1,n$,p$,a$,c$
1040 PRINT
1050 INPUT "on continue/RETURN   on arr)te/touche quelconque ";ch$
1060 IF CH$="" THEN 990
1070 RETURN
1080 REM CREATION
1090  OPEN "o",#1,"LYCEE
1100 GOSUB 970
1110 CLOSE
1120 RETURN
```

Commentaires

Programme simple ; on aurait pu simplifier avec des sous-programmes. Pour le tri par classe, on trie d'abord tout le lycée (lignes 690 à 730) puis on choisit la classe. Les recherches diverses montrent comment éviter l'emploi de plusieurs sous-programmes spécialisés.

Ex n° 2

```
10 TRAIT$=STRING$(80,"*")
20 PRINT "1 CREATION DU FICHIER STOCK
30 PRINT "2 CREATION DU FICHIER FOURNISSEUR
40 PRINT "3 AJOUT AU STOCK
50 PRINT "4 AJOUT AU FICHIER FOURNISSEUR
60 PRINT "5 EFFACEMENT AU SOTCK
70 PRINT "6 EFFACEMENT DU FICHIER FOURNISSEUR
80 PRINT "7 LECTURE DU FICHIER STOCK
90 PRINT "8 LECTURE DU FICHIER FOURNISSEUR
100 PRINT "9 VENTE
110 PRINT "0 COMMANDE
120 PRINT
130 INPUT "entrez votre choix ";A$
140 IF A$="1" THEN GOSUB 250
150 IF A$="2" THEN GOSUB 300
160 IF A$="3" THEN GOSUB 350
170 IF A$="4" THEN GOSUB 480
180 IF A$="5" THEN GOSUB 610
190 IF A$="6" THEN GOSUB 740
200 IF A$="7" THEN GOSUB 880
210 IF A$="8" THEN GOSUB 1020
220 IF A$="9" THEN GOSUB 1100
230 IF A$="0" THEN GOSUB 1260
240 GOTO 20
245 REM
250 REM -------------- CREATION DU FICHIER STOCK-------------
255 REM
260 OPEN "o",#1,"stock"
270 GOSUB 1810
280 CLOSE #1
290 RETURN
295 REM
300 REM -----------CREATION DU FICHIER FOURNISSEUR-----------
305 REM
310 OPEN "O",#1,"four"
320 GOSUB 1920
330 CLOSE #1
340 RETURN
345 REM
350 REM ---------------- AJOUT AU STOCK----------------------
355 REM
360 OPEN "i",#2,"STOCK"
370 OPEN "o",#1,"STOCK2"
380 WHILE NOT EOF(2)
390 INPUT #2,D$,S,sa,CA,PU,NF
400  WRITE #1,D$,S,sa,CA,PU,NF
410  WEND
420 CLOSE #2
430 GOSUB 1810
440 CLOSE #1
450 KILL "stock
460 NAME "STOCK2" AS "STOCK"
470 RETURN
475 REM
480 REM --------- AJOUT AU FICHIER FOURNISSEUR------------
485 REM
490 OPEN "i",#2,"FOUR
500 OPEN "o",#1,"FOUR2
```

```
510 WHILE NOT EOF(2)
520 INPUT #2,NF$,AF$
530 WRITE #1,NF$,AF$
540 WEND
550 CLOSE #2
560  GOSUB 1920
570 CLOSE #1
580 KILL "FOUR
590 NAME "FOUR2" AS "FOUR
600 RETURN
605 REM
610 REM -------- EFFACEMENT DU STOCK--------------
615 REM
620 INPUT "entrez le nom du produit à oter du fichier";NP$
630 NP$=UPPER$(NP$)
640 OPEN "i",#1,"STOCK"
650 OPEN "o",#2,"STOCK2"
660 WHILE NOT EOF(1)
670 INPUT #1,D$,S,sa,CA,PU,NF
680  IF NP$<> D$ THEN  WRITE #2,D$,S,sa,CA,PU,NF
690  WEND
700 CLOSE
710 KILL "Stock
720 NAME "STOCK2" AS "STOCK
730 RETURN
735 REM
740 REM ------- EFFACEMENT FICHIER FOURNISSEUR--------
745 REM
750 INPUT "entrez lz fournisseur à effacer du fichier ";NEF$
760 NEF$=UPPER$(nef$)
770 OPEN "i",#1,"FOUR
780 OPEN "o",#2,"FOUR2
790 WHILE NOT EOF(1)
800 INPUT #1,NF$,AF$
810  IF NEF$<> NF$ THEN WRITE #2,NF$,AF$
820 WEND
830 CLOSE #1
840 CLOSE #2
850 KILL "FOUR
860 NAME "FOUR2" AS "FOUR
870 RETURN
875 REM
880 REM ------- LECTURE SEQUENTIELLE DU FICHIER--------
885 REM
890 PRINT
900 PRINT trait$
910 PRINT "désignation";TAB(14);"stock initial";TAB(28);"cote
alerte";TAB(40)"stock actuek";;TAB(60);"prix HT";TAB(75)"fournisseur"
920 PRINT
930 OPEN "i",#1,"STOCK
940 WHILE NOT EOF(1)
950 INPUT #1,D$,S,SA,CA,PU,NF
960 GOSUB 2000
970 WEND
980 CLOSE
990 PRINT
1000 PRINT TRAIT$
1010 RETURN
1015 REM
1020 REM ------------ LECTURE FICHIER FOURNISSEUR----------
1025 REM
1030 OPEN "i",#2, "FOUR
```

```
1040 WHILE NOT EOF(2)
1050 INPUT #2,NF$,AF$
1060 PRINT NF$;TAB(30);AF$
1070 WEND
1080 CLOSE #2
1090 RETURN
1095 REM
1100 REM ------------------------- VENTE-------------------
1105 REM
1110 INPUT "entrez le produit ";p$
1120 INPUT "entrez la quantité";Q
1130 P$=UPPER$(P$)
1140 OPEN "i",#1,"STOCK
1150 OPEN "o",#2,"STOCK2"
1160 WHILE NOT EOF(1)
1170 INPUT #1,D$,S,sa,CA,PU,NF
1180 IF P$= D$ AND Q > SA THEN PRINT "erreur: la quantité demandée est
supérieure à la quantité en stock":GOTO 1210
1190  IF P$=D$ THEN SA=SA - Q
1200 WRITE #2,D$,S,sa,CA,PU,NF
1210 WEND
1220 CLOSE
1230 KILL "STOCK
1240 NAME "STOCK2" AS "stock
1250 RETURN
1260 REM --------------COMMANDE AUTOMATIQUE ---------------------------
1270 V=1
1280 DIM  F$(50), VA(50,4)
1290 OPEN "i",#1,"STOCK
1300 WHILE NOT EOF(1)
1310 INPUT #1,D$,S,sa,CA,PU,NF
1320  IF SA >= S  THEN 1390
1330 f$(V)=D$
1340 VA(V,1)=S
1350 VA(V,2)=SA
1360 VA(V,3)=NF
1370 VA(V,4)=PU
1380 V=V+1
1390 WEND
1400 CLOSE #1
1410 OPEN "i",#1,"FOUR
1420 NUF=1
1430 TOTAL=0
1440 WHILE NOT EOF(1)
1450  INPUT #1,Nf$,AF$
1460 PRINT "---------------- COMMANDE--------------------
1470 PRINT:PRINT:PRINT:PRINT:PRINT
1480 PRINT "société TRUC & MUCHE
1490 PRINT "rue des HAURDINNATEURRS
1500 PRINT
1510 PRINT " votre nom .....";NF$
1520 PRINT " votre adresse.....";AF$
1530 PRINT
1540 PRINT "                    Monsieur,
1550 PRINT
1560 PRINT "         Veuillez trouver ci-dessous la commande suivante:
1570 PRINT
1580 PRINT
1590 PRINT " désignation";TAB(30);"quantité       prix HT     total
1600 PRINT
1610  FOR I=1 TO 50
```

```
1620  IF NUF <> VA(I,3) THEN 1680
1630 Q=VA(i,1)-va(i,2)
1640 P=VA(i,4)
1650 T=Q*P
1660 TOTAL=TOTAL +T
1670 PRINT f$(i);TAB(30) USING "#####      #######.##  ######.##  ";Q;P;T
1680 NEXT
1690 PRINT
1700 PRINT " Montant de la commande..... ";total
1710 PRINT:PRINT:PRINT:PRINT:PRINT
1720 PRINT"************************************************************
1730 PRINT:PRINT:PRINT:PRINT:PRINT
1740 NUF=NUF+1
1750 TOTAL=0
1760 INPUT "entrez RETURN pour continuer ";C$:IF c$="" THEN
1770 WEND
1780 ERASE F$,VA
1790 CLOSE
1800 RETURN
1805 REM
1810 REM ----------- SOUS_PROGRAMME D ENTREE DES DONNEES STOCK-------
1815 REM
1820 INPUT "désignation du produit ";D$
1830 D$=UPPER$(d$)
1840 INPUT "stock initial ";S
1850 INPUT "cote d'alerte"; CA
1860 INPUT "Prix unitaire HT";PU
1870 INPUT "N( Du fournisseur";NF
1880 WRITE #1, D$,S,S,CA,PU,NF
1890 INPUT " RETURN/ ON CONTINUE AUTRE TOUCHE/ FIN ";C$
1900 IF C$="" THEN 1820
1910 RETURN
1915 REM
1920 REM --------- SOUS_PROGRAMME D ENTREE DES FOURNISSEURS-------
1925 REM
1930 INPUT "nom du fournisseur ";NF$
1940 NF$=UPPER$(NF$)
1950 INPUT "Adresse ";AF$
1960 WRITE #1,nf$,AF$
1970 INPUT " RETURN/ ON CONTINUE AUTRE TOUCHE/ FIN ";C$
1980 IF C$="" THEN 1930
1990 RETURN
1995 REM
2000 REM --------------------- AFFICHAGE--------------------------
2005 REM
2010 PRINT D$;TAB(15)USING " #####        #####       #####
#####.##        ####";s;ca;sa;pu;nf;
2020 IF SA < CA THEN PRINT " <<>>" ELSE PRINT
2030 RETURN
```

Commentaires :

Rien d'extraordinaire dans ce programme ; il aurait pu être simplifié en usant plus souvent de sous-programmes, particulièrement à l'ouverture et à la fermeture des fichiers. On remarquera cependant la commande automatique. Elle utilise des tableaux, F$ et VA, qui servent à recevoir les valeurs du stock ; il est plus facile de traiter les don-

nées d'un fichier séquentiel en les chargeant d'abord dans des tableaux. Ici, on entre en RAM les valeurs inférieures à la côte d'alerte, ainsi que les valeurs correspondantes, puis la commande se fait en regroupant dans une même lettre les produits manquants du même fournisseur. Les lignes 1460 à 1600 servent à écrire l'en-tête de la lettre de commande, et l'édition se fait en prenant pour base de calcul le stock initial et le stock actuel. Par ailleurs, la lecture du stock permet de visualiser les produits en rupture : c'est en ligne 2020.

9.2. LES FICHIERS A ACCÈS DIRECT

Principes généraux :

On comprend très vite les inconvénients des fichiers à accès séquentiel : plus ils sont longs et plus la durée des manipulations augmente. On va donc leur préférer les fichiers à accès direct. Le principe de base est très simple.

Nous avons vu que le fichier à accès séquentiel est un ensemble d'enregistrements dont la longueur est variable ; ainsi, le premier enregistrement peut avoir 130 caractères, le second 50, le troisième 200 etc. Dans ce cas, on se souvient que les enregistrements sont stockés sur la disquette tout bonnement les uns derrière les autres : c'est d'ailleurs là tout le problème, et on va mieux comprendre pourquoi justement en analysant l'accès direct.

L'accès direct demande au programmeur de prévoir la longueur de chaque enregistrement avant écriture sur disque ce qui fait que, ne pouvant prévoir la longueur de chaque enregistrement, on va plus simplement prévoir une longueur fixe, identique à tous les enregistrements. Cette longueur sera donc à la discrétion du programmeur, dans les limites précises, et supposée adaptée dans tous les cas ; ainsi, si on fait un fichier de noms propres, on fixera une longueur de 20 caractères, en supposant que 20 caractères suffiront : si un nom a moins de 20 caractères, pas de problème car les espaces restants seront comblés de blancs ; si le nom a plus de 20 caractères, les caractères en trop seront ignorés par le système.

Ainsi, le fichier pourra être :

```
DURAND!!!!!!!!!!!!!!!/BIHAN!!!!!!!!!!!!!!!!!/DUPONT DE LA TOUR D/

|                    |                    |
1er car.             21ème Car.           41ème car.
```

On voit que si DURAND et BIHAN tiennent largement dans l'espace réservé, un nom tel que DUPONT DE LA TOUR D'AUVERGNE, dont la longueur excède 20 caractères est tronqué : c'est un léger inconvénient à vrai dire ; tous les noms propres ne sont pas si longs, il faut donc accepter quelques petits sacrifices. Bien sûr, on pourrait réserver 40 caractères pour que DUPONT DE LA TOUR D'AUVERGNE entre en entier dans le fichier. Mais il ne faut pas perdre de vue le fait que les espaces non occupés des enregistrements plus courts sont quand même réservés sur le disque : cela serait donc de la place gaspillée.

On remarquera que les enregistrements peuvent être imaginés les uns derrière les autres, comme disposés séquentiellement : ils peuvent donc être numérotés 1, 2, 3 etc, sans oublier le fait que le système connaît le lieu sur la disquette du premier enregistrement.

L'intérêt de réserver une longueur fixe avant utilisation s'explique donc : si on désire lire le n-ième enregistrement, le système calcule l'emplacement sur le disque de cet enregistrement en multipliant n – 1 par la longueur de l'enregistrement et y accède directement (En fait, tout se passe pour l'utilisateur comme cela, bien que la réalité soit un peu différente), en placant la tête de lecture au bon endroit :

Si on veut lire le troisième, n = 3, donc :

(3 – 1)*20 = 40 caractères : le 3ème enregistrement commence sur la disquette après le 40ème caractère, ainsi que le dessin précédent le montre (Ce calcul est fait par le système et non par le programmeur). Cette faculté d'accéder directement à l'enregistrement désiré permet donc une vitesse accrue du traitement de l'information sur disque. Mais, outre ceci, un autre facteur très intéressant est que l'on peut modifier une donnée ou l'effacer sans pour autant recopier le fichier dans un autre comme nous le faisions en accès séquentiel. S'il fallait, dans ce dernier, toute une série de manipulations exigée par le mode propre d'écriture d'enregistrements de longueur différente, il n'y a plus ce problème en accès direct, puisqu'il suffira de modifier ou d'effacer uniquement et directement l'enregistrement en question.

9.2.1. La création et l'utilisation

OPEN "R", FIELD, PUT #, LSET, RSET

Pour créer ET utiliser un fichier à accès direct, il suffit de l'ouvrir avec OPEN "R". La lettre "R" vient de Random (random access, accès aléatoire, car on utilise aussi l'expression "fichier à accès aléatoire",

pour signifier l'accès à n'importe quelle donnée). Il faut comme en accès séquentiel réserver le numéro de fichier (de buffer) avec #, nommer le fichier à créer ou à utiliser et, optionnellement, indiquer la longueur d'un enregistrement : la longueur par défaut est de 128 caractères, on peut donc omettre de le préciser, ou bien on indique la longueur réelle de l'enregistrement. Pour illustrer avec un exemple, nous allons créer un fichier téléphone, nommé "TEL" (pour changer !) ; il faut au préalable définir l'enregistrement ET définir les champs composant cet enregistrement, comme en accès séquentiel, mais en précisant cette fois-ci la longueur des champs, lesquels, additionnés fourniront la longueur totale de l'enregistrement ; nous dirons que :

— on réserve 20 caractères pour le nom, NOM$;
— on réserve 20 caractères pour le prénom, PRENOM$;
— on réserve 40 caractères pour l'adresse, ADRESSE$;
— on réserve 20 caractères pour la ville, VILLE$;
— on réserve 14 caractères pour le numéro de tel, TEL$.

Un enregistrement mesure donc :

20 + 20 + 40 + 20 + 14 = 114 caractères.

Pour ouvrir le fichier :

```
100 OPEN "R", #1, "TEL", 114
```

Pour déclarer les champs :

```
110 FIELD #1, 20 AS NOM$, 20 AS PRENOM$, 40 AS ADRESSE$, 20 AS
    VILLE$, 14 AS TEL$
```

On note le FIELD (de field, champ) et la déclaration de la longueur de chaque champ, AS (de as, comme, en tant que).

Ouvrons une parenthèse. On avait parlé, lors du passage sur les fichiers à accès séquentiel, d'une mémoire-tampon. Il faut savoir que, dès que le Basic rencontre l'instruction OPEN, il réserve cet espace, nommé également "buffer", en mémoire et le numérote, ici avec le nombre suivant le signe #. A quoi sert cet espace ? Cet espace sert de lieu transitoire de passage et est obligatoire, lors des opérations de lecture et d'écriture. Quand on voudra écrire par exemple, il faudra d'abord entrer les valeurs à écrire dans des variables, puis recopier ces variables dans le buffer : ce dernier est organisé selon la présentation définie justement grâce à FIELD # 1. Dans notre exemple, on a défini un buffer de numéro 1 (car on peut utiliser simultanément plusieurs fichiers, donc de numéro 1, 2 etc), qui est divisé en plusieurs

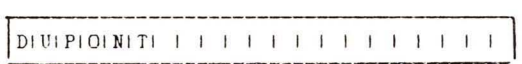
morceaux, les champs, eux-mêmes repérés par un nom particulier, NOM$, PRENOM$ etc. Quand on voudra emplir ce buffer, il faudra au préalable emplir des variables de noms différents :

```
120 INPUT "entrez le nom ";N$
130 INPUT "entrez le prénom";P$
140 INPUT "entrez l'adresse ";A$
150 INPUT "entrez la ville ";V$
160 INPUT "entrez le téléphone";T$
```

Puis il faut transférer ces valeurs dans la mémoire-tampon en utilisant le terme LSET (de left set, positionner, cadrer à gauche) :

```
170 LSET NOM$=N$
180 LSET PRENOM$=P$
190 LSET ADRESSE$=A$
200 LSET VILLE$=V$
210 LSET TEL$=T$
```

Pourquoi ce terme LSET ? On peut écrire une valeur à gauche dans une mémoire réservée ainsi :



ou bien à droite, avec RSET (de right set, positionner, cadrer à droite) :

Puis il faut transférer le contenu du buffer sur le disque avec PUT (de put, mettre) :

```
220 PUT #1
```

On peut indiquer le numéro de l'enregistrement avec :

220 PUT #1, NUM

il aurait fallu pour cela déterminer au départ la valeur de NUM, puis incrémenter cette dernière à chaque écriture ; la première façon est plus simple, car le système détermine seul le numéro de l'enregistrement ; nous verrons comment utiliser la seconde façon ultérieurement.

Pour entrer d'autres valeurs, on se contente ici de faire une boucle ou bien de finir :

```
230 INPUT " On cont/ RETURN  on arrete/ une touche "; CHOIX$
240 IF CHOIX$="" THEN 120
250  CLOSE #1
```

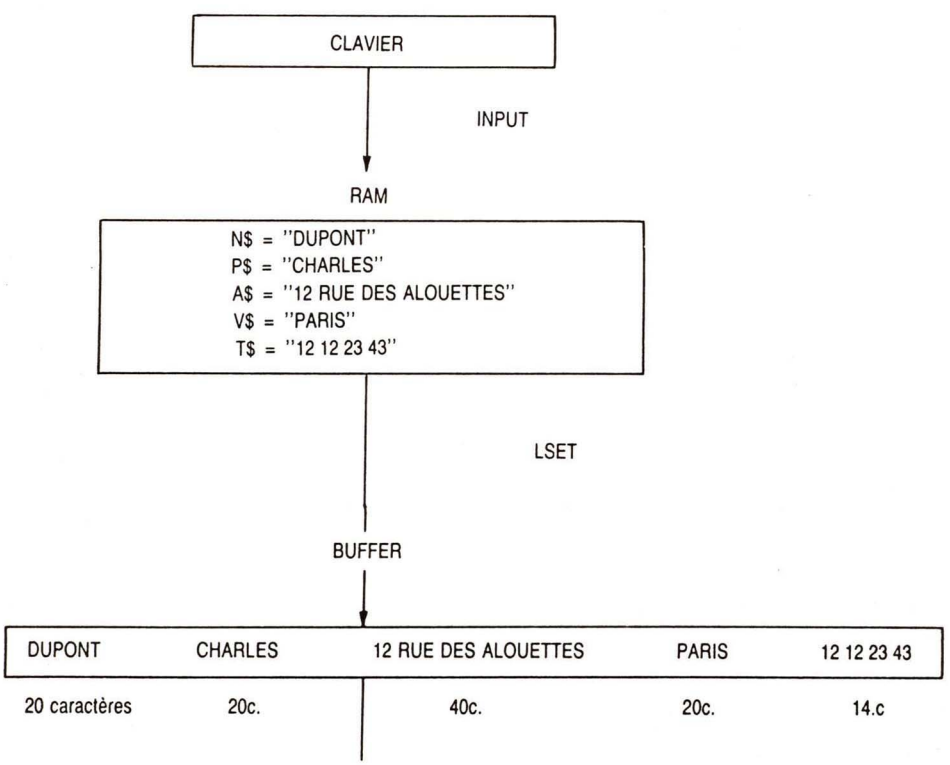
Ainsi, créer un fichier à accès direct se résume à :

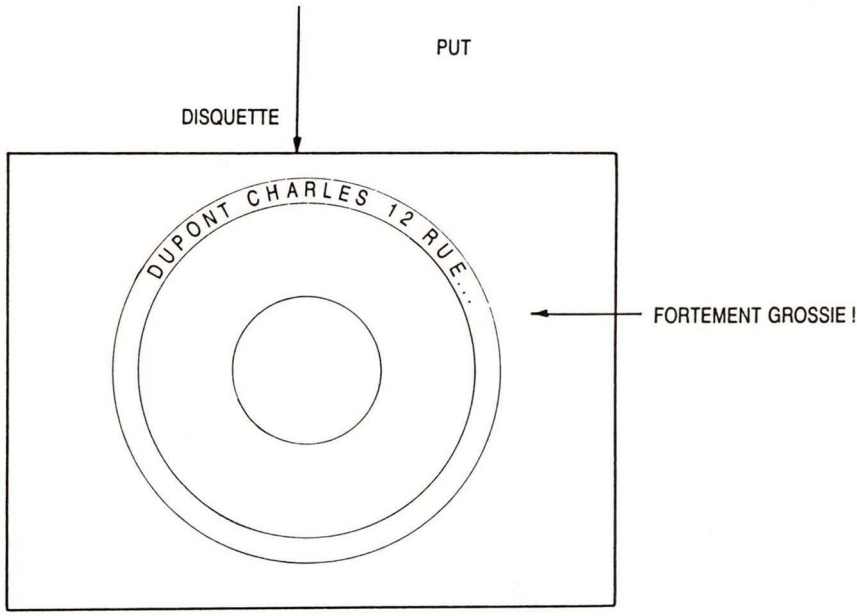
— ouvrir le fichier en déclarant son numéro de buffer, son nom, la longueur de l'enregistrement et celle des champs nommés
— entrer des valeurs dans des variables
— transférer ces variables dans une mémoire spéciale
— transférer le contenu de cette mémoire sur le disque
— fermer le fichier

Par exemple, en supposant que l'on désire entrer sur disque la personne suivante :

— nom : DUPONT
— prénom : CHARLES
— adresse : 12 RUE DES ALOUETTES
— ville : PARIS
— téléphone : 12 12 23 43

nous aurons le schéma :





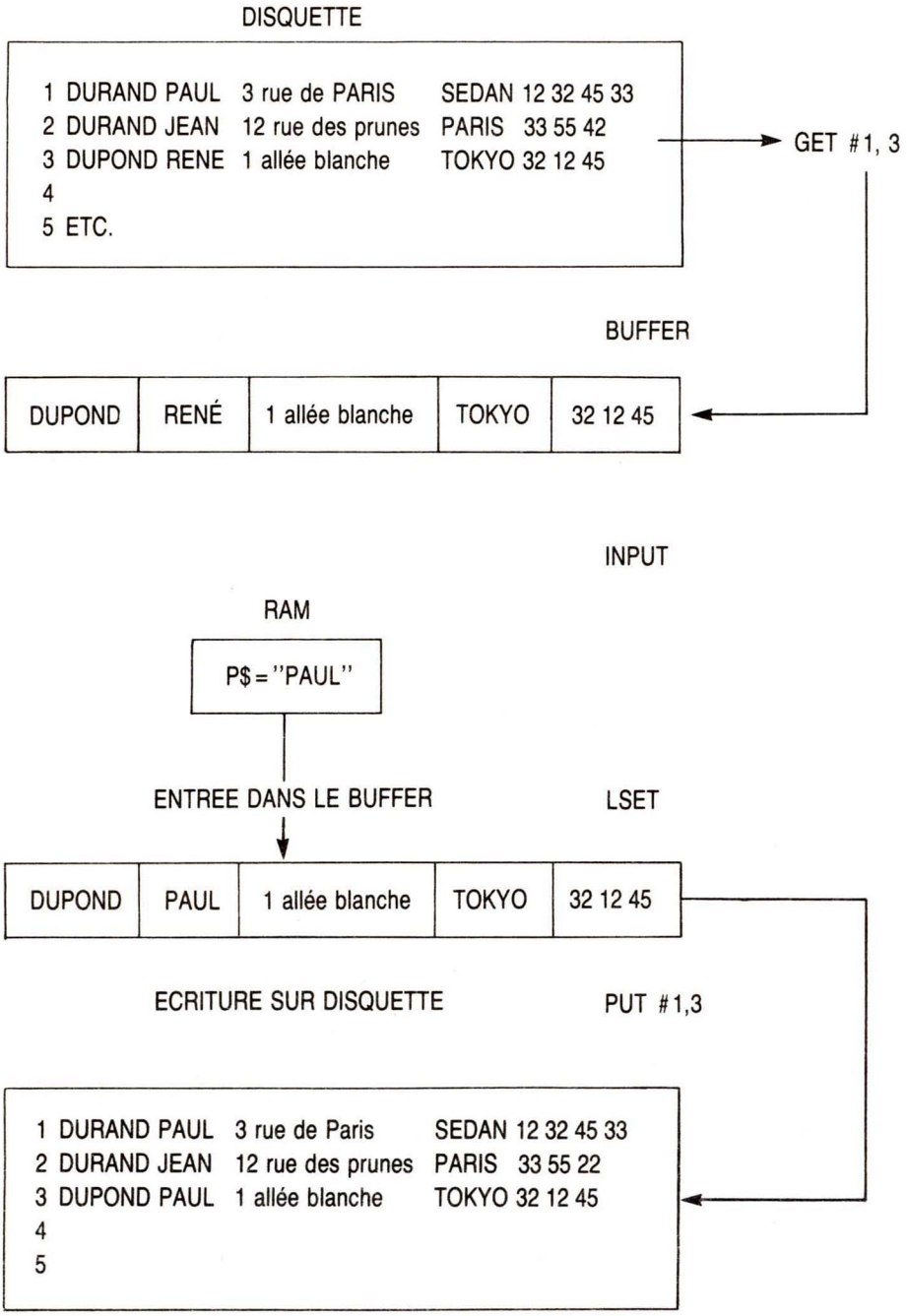
Il ne suffit pas de créer le fichier, encore faut-il le relire ; comme un enregistrement est repéré par son numéro, il suffit de rappeler ce numéro avec GET (de get, obtenir), après l'avoir ouvert de nouveau :

```
INPUT "entrez le numéro recherché "; NUM
GET #1, NUM
PRINT NOM$, PRENOM$,ADRESSE$,VILLE$,TEL$
```

On note qu'il ne faut pas prendre en compte les variables-champs pour relire un enregistrement.

Pour modifier un enregistrement, il suffit de le rappeler avec GET et de changer le champ ou les champs désirés. Il faut bien comprendre que, en rappelant un enregistrement, on le transfère dans le buffer correspondant avec tous ses champs ; on peut donc réécrire un champ sans toucher aux autres du buffer puis recopier l'ensemble sur disque :

Ainsi, en supposant corriger le prénom dans le troisième enregistrement pour le remplacer par PAUL, on va chercher l'enregistrement numéro 3 sur la disquette :




```
INPUT "entrez le numéro de l'enregistrement à modifier ";N
GET #1,N
INPUT "entrez le nouveau prénom ";P$
LSET PRENOM$=P$
PUT #1,N
```

Il faut remarquer qu'ici, l'exemple est choisi pour changer uniquement le prénom. Il faudra donc prévoir dans un tel fichier que l'on puisse changer les autres champs.

Une instruction peut être utilisée pour connaître le dernier enregistrement lu ou écrit ; il suffit de faire :

PRINT LOC (n° de fichier)

9.2.2. Les nombres et leur conversion

Nous avons vu dans le chapitre consacré au traitement de chaînes de caractères comment transformer un nombre en chaîne et inversement, avec les fonctions VAL et STR$. Cette aptitude du Basic est primordiale dans la gestion de fichiers à accès direct et indexé. En effet, s'il n'était pas nécessaire de le faire en accès séquentiel, cela le devient maintenant. On a remarqué que l'objet de l'accès direct est de mémoriser sur un disque des informations essentiellement alphanumériques : pourquoi alphanumériques ? Parce qu'il faut connaître au préalable la longueur des champs ; or, il n'est pas possible de prévoir la longueur, en nombre de caractères, de telle ou telle valeur numérique : le nombre 3 a un caractère ; le nombre 124 en a trois. Nous sommes donc obligés de convertir une variable ou une constante numérique en valeur alphanumérique ; nous savons le faire avec les deux instructions citées, VAL et STR$. Leur inconvénient est qu'elles provoquent une plus grande consommation d'octets, donc de place sur le disque. On va donc leur préférer d'autres fonctions, qui auront le même usage sans le même inconvénient, et ces fonctions sont multiples, selon la qualité de la valeur à conserver. Cette qualité dépend du nombre :

— entier
— réel simple précision
— réel double précision.

Cette différence est simple :

a = 10 est une valeur entière,

a = 10.34 est une valeur simple précision et

a = 123243343,324143 est une valeur double précision : ces trois valeurs vont donc être mémorisées de manière différente.

Une valeur entière se code sur deux octets avec l'ordre MKI$ (de make from integer, créer à partir d'un entier), une valeur réelle simple précision avec MKS$ et se code sur quatre octets (de make from single, créer à partir d'une valeur simple) et une valeur réelle double pré-

cision avec MKD$ et se code sur huit octets (de make from double, créer à partir d'une valeur double).

REMARQUE : une valeur entière varie de – 32766 à 32767 car ce sont les limites des valeurs codables sur deux octets.

On utilise ainsi ces fonctions :

soit un fichier dont un champ est déclaré comme ceci :

FIELD #1, 4 AS PRIX$

Pour entrer une valeur dans PRIX$, on peut faire :

INPUT ''entrez le prix'' ;P

puis traduire le nombre P :

LSET PRIX$ = MKS$(P)

Il faut cependant, à la lecture du fichier et si on veut réutiliser les valeurs numériques, les retransformer en nombres ; cette action se programme en fonction du codage précédent. Si on a utilisé :

MKI$, on se servira de la fonction CVI (de convert to integer)

MKS$, on se servira de la fonction CVS (de convert to single)

MKD$, on se servira de la fonction CVD (de convert to double)

Ainsi, pour reprendre l'exemple précédent, on dira, après GET :

Prix = CVS (PRIX$)

9.2.3. Structures multiples de la mémoire-tampon

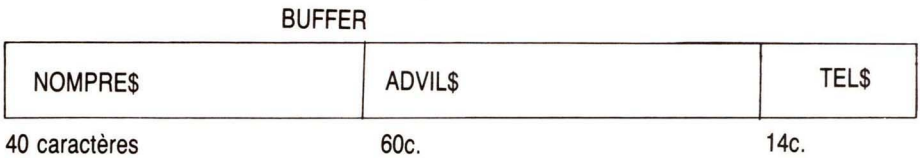
Un aspect fort intéressant de la mémoire-tampon est le ''saucissonnage'' possible. On sait que la mémoire en question est simplement un ensemble numéroté de caractères, et que cet ensemble est découpé en tranches nommées ''champs'' de longueur variable. Par exemple, la mémoire organisée avec le programme cité précédemment est ainsi :

BUFFER

NOM$	PRENOM$	ADRESSE$	VILLE$	TEL$
20 caractères	20c.	40c.	20c.	14c.

Ce découpage est défini par l'instruction FIELD suivie de la longueur et des noms des champs : tant qu'aucune autre indication sur

ce buffer ainsi découpé n'est pas précisé, le système, à chaque fois que le programme lira NOM$, PRENOM$, ADRESSE$ etc comprendra qu'il s'agit de variables du buffer. Mais rien n'empêche le programmeur de découper le buffer selon d'autres champs, par exemple en regroupant le nom et le prénom pour n'en faire qu'une seule variable, ainsi que l'adresse et la ville :



Ainsi avons-nous maintenant deux formats en mémoire utilisables et déclarés de cette manière :

```
FIELD #1, 20 AS NOM$,20 AS PRENOM$, 40 AS ADRESSE$,20 AS VILLE$,14
AS TEL$
FIELD #1, 40 AS NOMPRE$, 60 AS ADVIL$ , 14 AS TEL$
```

Comment l'ordinateur s'y retrouve ? Simplement avec le nom de la variable qu'il lit au moment de l'exécution ; quant il lit NOM$, il sait qu'il doit faire référence à une chaîne de longueur 20 située au début du buffer ; quand il lit ADVIL$, il sait qu'il doit faire référence à une chaîne commençant au 41-ème caractère et de longueur 60. En fait, le buffer est toujours le même, seule change l'appellation des champs.

Cette aptitude aux multiples appellations peut être particulièrement intéressante ; en effet, on peut par exemple utiliser un enregistrement à des fins autres que celles prévues : on peut stocker dans le premier enregistrement un nombre représentant le nombre d'enregistrements du fichier, ou bien la date de création, le nom de l'utilisateur etc ; comme ces valeurs n'ont rien à voir avec les enregistrements ''normaux'' du fichier, elles seront donc nécessairement différentes dans leur structure et donc utilisables autrement ; un FIELD particulier pourra leur être attribué. Pour reprendre l'exemple précédent, on peut prévoir un format spécial destiné à éffacer tous les champs d'un coup :

```
FIELD #1, 114 AS RIEN$
```

A l'utilisation d'un tel buffer, on pourra dire, par exemple :

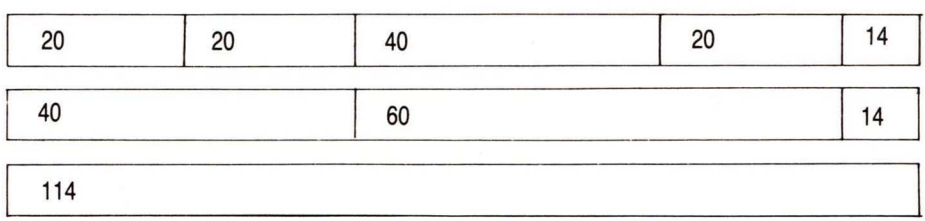
```
INPUT "entrez le numéro de l'enregistrement à effacer ";N
LSET RIEN$=STRING$(114," ")
PUT #1,N
```

Ici, on a écrit dans le buffer 114 espaces, autrement dit rien ; puis on transfère ces 114 ''riens'' sur le disque au numéro convenu et ainsi 114 espaces recouvrent les 114 caractères composant l'enregistrement numéro N : l'enregistrement en question est donc effacé du fichier.

Nous avons donc fait :

```
FIELD #1, 20 AS NOM$,20 AS PRENOM$, 40 AS ADRESSE$,20 AS VILLE$,14
AS TEL$
FIELD #1, 40 AS NOMPRE$, 60 AS ADVIL$ , 14 AS TEL$
FIELD #1, 114 AS RIEN$
```

Ce qui crée trois découpages possibles ;



9.2.4. Considérations sur les fichiers :

Quand on utilise le Basic, on peut faire tourner ensemble au plus trois fichiers, et la longueur par défaut d'un enregistrement est de 128 caractères.

Il peut cependant s'avérer nécessaire d'utiliser plus de trois fichiers composés d'enregistrements plus longs. Au moment de charger le Basic, au lieu de faire :

A>BASIC → RETURN

On fera :

A>BASIC /F : nombre de fichiers désiré /S : taille de l'enregistrement

(F, de file, fichier et S de size, la taille).

EXERCICES

Faire le programme permettant la gestion d'un stock. Il n'y aura qu'un seul fichier. Le nombre d'enregistrements sera stocké en mémoire.

```
10 REM --------- FICHIER A ACCES DIRECT ----------
20 REM
30 REM ------      gestion      de     stock     ----------
40 REM
50 REM
60 INPUT "nom du fichier stock ";NOMStock$
70 OPEN "r",#1,NOMSTOCK$,40
80 FIELD #1,2 AS NUM$,38 AS EFf$
90 FIELD #1, 40  AS RIEN$
100 FIELD #1,20  AS DESIG$,4 AS NUF$,4 AS SI$, 4 AS CA$, 4 AS SA$,4 AS PU$
110 GET #1,1
120 NUM=CVI(NUM$)
130 IF NUM=0 THEN NUM=1
140 PRINT " nombre d'enregistrements= ";NUM
150 REM
160 REM -------------- MENU PRINCIPAL ----------------
170 REM
180 PRINT "1 INITIALISATION DU FICHIER STOCK
190 PRINT "2 AJOUT AU STOCK
200 PRINT "3 EFFACEMENT AU SOTCK
210 PRINT "4 MODIFICATION
220 PRINT "5 RECHERCHE
230 PRINT "6 LECTURE
240 PRINT "7 VENTE
250 PRINT "8 FERMETURE FINALE
260 PRINT
270 INPUT "entrez votre choix ";A
280 ON  A GOSUB  320, 410, 590 ,690, 990, 1060 ,1170, 1280
290 GOTO 180
300 REM
310 REM ----- INITIALISATION DU FICHIER STOCK ---------
320 REM
330 LSET NUM$=MKI$(0):LSET EFF$=STRING$(28," ")
340 PUT #1,1
350 LSET RIEN$=STRING$(30," ")
360 FOR I=2 TO 150
370 PUT #1,I
380 NEXT
390 RETURN
400 REM
410 REM ------ ENTREE D UN PRODUIT DANS LE FICHIER STOCK---
420 NUM=NUM+1
430 INPUT "designation du produit.....";DE$
440 INPUT "numero du fournisseur.....";NUF
450 INPUT "stock initial.....";si
460 INPUT "cote d 'alerte.....";ca
470 INPUT "prix unitaire hors-taxes.....";PU
480 LSET DESIG$=DE$
490 LSET NUF$=MKS$(NUF)
500 LSET SI$=MKS$(SI)
510 LSET CA$=MKS$(CA)
520 LSET SA$=MKS$(SI)
530 LSET PU$=MKS$(PU)
540 PUT #1,NUM
550 INPUT "on cont/RETURN on arrete/ touche quelconque ";C$
560 IF C$="" THEN 420
570 RETURN
580 REM
590 REM  ------------------- EFFACEMENT----------------
600 REM
610 INPUT "numero de l'enregistrement a effacer ";NE
```

```
620 GET #1,NUM
630 PUT #1,NE
640 LSET RIEN$=STRING$(40," ")
650 PUT #1,NUM
660 NUM=NUM-1
670 RETURN
680 REM
690 REM ------------------- MODIFICATION---------------
700 REM
710 INPUT "entrez le numero a modifier";i
720 GET #1,I
730 GOSUB 1380
740 PRINT " 1 DESIGNATION  2 NUM FOURNISSEUR  3 STOCK INITIAL  4 COTE
ALERTE  5 STOCK ACTUEL 6 PUHT    "
750 INPUT "entrez le numero de la rubrique  a modifier ";rm
760 ON RM GOSUB 800, 830, 860,890,920,950
770 PUT #1,I
780 PRINT "enregistrement modifié ": GOSUB 1380:PRINT
790 RETURN
800 INPUT "entrez la nouvelle désignation ";de$
810 LSET DESIG$=DE$
820 RETURN
830 INPUT "entrez le numéro du fournisseur ";NF
840 LSET NUF$=MKS$(NF)
850 RETURN
860 INPUT "entrez la valeur du stock initial ";SI
870 LSET SI$=MKS$(SI)
880 RETURN
890 INPUT "entrez la valeur de la cote d'alerte ";ca
900 LSET CA$=MKS$(CA)
910 RETURN
920 INPUT "entrez le Stock actuel ";sa
930 LSET SA$=MKS$(SA)
940 RETURN
950 INPUT "entrez le prix unitaire hors taxes ";PU
960 LSET PU$=MKS$(PU)
970 RETURN
980 REM
990 REM ----------------------RECHERCHE--------------------
1000 REM
1010 INPUT "numero @ lire "; I
1020 GET #1, I
1030 GOSUB 1380
1040 RETURN
1050 REM
1060 REM --------------- LECTURE SEQUENTIELLE ---------------
1070 REM
1080 INPUT "0 SUR L'ECRAN    /    1 SUR L 'IMPRIMANTE ";EI
1090 IF EI = 0 THEN PRINT "NUM  DESIGNATION      N. FOUR  SK INITIAL  CO
ALERTE  STOCK ACTUEL   P U H.T  ALERTE":PRINT   ELSE LPRINT "num
désignation          F.  STC Init C.Aler STC Act. PUHT   Alerte":LPRINT
1100 FOR I=2 TO NUM
1110 GET #1,I
1120 IF EI = 0 THEN  GOSUB 1380 ELSE  GOSUB 1440
1130 NEXT
1140 PRINT
1150 RETURN
1160 REM
1170 REM --------------------VENTE ------------------
1180 REM
1190 INPUT "designation de l'objet ";ob$
```

```
1200 INPUT "quantite vendue ";qv
1210 FOR i=2 TO NUM
1220 GET #1,I
1230 IF  OB$=LEFT$(DESIG$,LEN(OB$)) AND  QV > CVS(SA$)  THEN PRINT
"quantité insuffisante ":GOSUB 1380:PRINT: RETURN
1240 IF  OB$=LEFT$(DESIG$,LEN(OB$)) THEN  GOSUB 1380:
reste=CVS(SA$)-QV:LSET SA$=MKS$(RESTE):PUT #1,I:GOSUB 1380:RETURN
1250 NEXT
1260 PRINT " le produit n'est pas en stock:vente impossible "
1270 RETURN
1280 REM
1290 REM --------------FERMETURE DU FICHIER -----------
1300 REM
1310 LSET NUM$=MKI$(NUM)
1320 PUT #1,1
1330 PRINT "suvegarde de ";num;"enreg"
1340 CLOSE
1350 PRINT "fermeture finale : merci de votre visite !
1360 END
1370 REM
1380 REM ----------SOUS PROGRAMME D AFFICHAGE----------
1390 REM
1400 PRINT  I;desig$;:PRINT  USING "####     ####       ####            ####
####.##   " ;CVS(NUF$);CVS(SI$);CVS(CA$);CVS(SA$);CVS(PU$);
1410 IF  CVS(SA$) < CVS(CA$) THEN PRINT "<<-->>" ELSE PRINT
1420 RETURN
1430 REM
1440 REM------ -----------IMPRIMANTE----------
1450 REM
1460 LPRINT  I;desig$;:LPRINT  USING "####  ####    ####    ####    ####.##
" ;CVS(NUF$);CVS(SI$);CVS(CA$);CVS(SA$);CVS(PU$);
1470 IF  CVS(SA$) < CVS(CA$) THEN LPRINT "<<-->>" ELSE LPRINT
1480 RETURN
```

Analyse de la vente : ligne 1230.

On voit que, effacer un enregistrement de n° NE, c'est mettre à sa place le dernier enregistrement, de n° NUM et effacer ce dernier du fichier (lignes 600-670).

9.3. LES FICHIERS A ACCÈS INDEXÉ

Intérêts et principes

On a vu avec l'accès direct qu'il est possible d'obtenir un enregistrement si l'on connaît son numéro, sinon nous sommes obligés de faire une recherche séquentielle. Pourtant, si on avait la liste triée par champs des enregistrements avec leur numéro d'emplacement sur la disquette, il serait plus rapide d'aller chercher le renseignement demandé. Or, il est parfaitement possible d'avoir cette liste : ou bien

en la créant ou bien en se servant de JETSAM, un gestionnaire de fichiers à accès indexé intégré dans le Basic de l'AMSTRAD PCW 8256.

Quel est l'intérêt d'une telle liste ?

Supposons avoir créé un fichier à accès direct traitant de clients référencés par leur nom, leur adresse et leur numéro de téléphone. Si nous cherchons le client "DUPOND" par son nom, nous sommes obligés de lire séquentiellement le fichier jusqu'à obtention du résultat. Par contre, si nous possédons la liste triée des clients, il nous suffira de chercher notre client parmi les noms dont la première lettre est un "D" puis de lire le numéro de l'enregistrement correspondant.

Pour illustrer d'un exemple plus courant la différence entre l'accès séquentiel, direct et indexé, regardons la manière que nous utilisons quand nous cherchons un mot dans cet ouvrage, par exemple le mot "JETSAM" :

Si nous lisons le livre en partant du premier mot de la première page, nous aboutirons obligatoirement au mot "JETSAM", mais nous aurons lu toutes les pages précédentes : c'est de l'accès séquentiel.

Si nous savons que le mot "JETSAM" est à la page 129 en haut à gauche : c'est de l'accès direct et c'est donc très rapide mais il fallait le savoir...

Si nous lisons le sommaire ou mieux l'index (le bien-nommé !) qui est la liste alphabétique des mots utilisés dans l'ouvrage, nous verrons en face de JETSAM le numéro de la page 129 ; c'est de l'accès indexé, et même précisément "séquentiel indexé", puisqu'il aura fallu lire séquentiellement l'index.

On comprend donc l'intérêt de posséder une table d'index.

Pour revenir à l'exemple précédent, nous dirons avoir cherché le mot "DUPOND" dans une table composée de deux éléments : la clé et l'index.

La clé est simplement le mot formant l'objet de la recherche : si on a créé un accès indexé sur le nom, on a constitué une liste de noms, ou une liste de "clés". Chercher "DUPOND" dans la table, c'est chercher si la clé "DUPOND" s'y trouve.

L'index est l'élément numérique indispensable correspondant à la clé et renvoyant à l'enregistrement complet : si "DUPOND" a le numéro 1282, cela signifie qu'il est enregistré sur la disquette à l'emplacement n°1282.

Par ailleurs, l'intérêt d'analyser un fichier à partir d'un index se justifie lorsque nous voulons la liste triée des enregistrements ; nous savons qu'un tri normal peut prendre beaucoup de temps, même en

utilisant les algorithmes de tri les plus rapides (comme QUICK SORT) et peut même être impossible à trier en mémoire vive (à cause du manque de place s'il y beaucoup de données). Le seul fait de trier sur une clé tout en gardant le numéro de l'enregistrement permet et la vitesse et le gain de place.

Pour étudier l'accès indexé, nous apprendrons d'abord à créer un index en mémoire vive, sans pour autant s'attarder sur cette méthode (il en existe plusieurs variantes, des plus simples au plus compliquées), puis nous étudierons l'accès indexé avec JETSAM.

9.3.1. L'accès indexé simulé en RAM :

Considérons le programme suivant :

```
10 DATA ZOLA,EMILE,30,HUGO
,VICTOR,18,RIMBAUD,ARTHUR,19,MAURIAC,FRANCOIS,24,BAZIN,HERVE,44,FOUCAULT,MI
CHEL,22,LEFORT,CLAUDE,44
20 N=7
30 DIM CLE$(N),INDEX(N)
40 REM
50 REM ------CREATION DU FICHIER --------------------
60 REM
70 OPEN "r",#1, "M:FICHIER",42
80 FIELD #1, 20 AS NOM$,20 AS PRENOM$,2 AS AGE$
90 FOR I=1 TO N
100 READ   NM$,PRE$,AGE
110 LSET NOM$=NM$
120 LSET PRENOM$=PRE$
130 LSET AGE$=MKI$(AGE)
140 PUT #1,I
150 NEXT
160 REM ------CONSTITUTION DU TABLEAU DE TRI----------
170 REM
180 INPUT "entrez le champ de tri ";VALEUR$
190 VALEUR$=UPPER$(VALEUR$)
200 FOR I=1 TO N
210 GET #1,I
220 IF VALEUR$="NOM" THEN CLE$(I) = NOM$
230 IF VALEUR$="PRENOM" THEN CLE$(I) = PRENOM$
240 IF VALEUR$="AGE" THEN CLE$(I) = AGE$
250 INDEX(I) = I
260 NEXT
270 REM
280 REM ----------------TRI INDEXE ------------------
290 REM
300 PRINT "tri index( sur ";VALEUR$
310 PRINT
320 FOR I=1 TO N-1
330 FOR J=1+I TO N
340 IF CLE$(J) <= CLE$(I) THEN SWAP CLE$(I) ,CLE$(J):SWAP INDEX(I),INDEX(J)
350 NEXT
360 NEXT
```

```
370 REM
380 REM --------LECTURE DU FICHIER TRIE --------------
390 REM
400 PRINT "n[ Enrg.  Nom          Pr(nom              Age"
410 PRINT
420 FOR I=1 TO N
430 GET #1,INDEX(I)
440 PRINT INDEX(I);TAB(4);NOM$;TAB(25);PRENOM$;TAB(46);CVI(AGE$)
450 NEXT
```

Disons tout de suite que ce programme ne présente qu'un intérêt pédagogique. Nous admettrons que nous faisons un fichier nommé "FICHIER" dont les enregistrements sont composés d'un nom, d'un prénom et d'un âge.

Les lignes 50-150 servent à entrer dans le fichier à accès direct les noms, prénoms et âges de la ligne 10 de la manière habituelle. Ce qui est intéressant commence en ligne 160, à la constitution des tableaux CLE$ et INDEX. La ligne 180 demande sur quelle clé voulons-nous l'index. La réponse détermine si nous le voulons sur le nom, le prénom ou l'âge et CLE$ et INDEX se remplissent séquentiellement du fichier ; en d'autres termes, on recopie un champ particulier des enregistrements (tous les noms, tous les prénoms ou tous les âges) dans le tableau CLE$ et le tableau INDEX se remplit du numéro d'enregistrement correspondant au champ : comme l'entrée du champ dans CLE$ est séquentielle, INDEX est simplement séquentiel.

Ainsi sont constitués en RAM deux tableaux puis les lignes 280 à 360 servent à trier les données du tableau CLE$ et la ligne 340 permet le tri des valeurs de INDEX. Or, ces valeurs triées en même temps que les valeurs de CLE$ représentent les numéros d'enregistrement de ces dernières. Quand le tri est fini, on peut lire le tableau CLE$ dans lequel sont triées les clés. Mais le plus important est dans le tableau INDEX : il contient maintenant les enregistrements classés par leur numéro ; il suffit, pour lire les clés triées ET l'enregistrement complet correspondant, d'aller chercher à partir de l'index INDEX les enregistrements sur disque.

Exemple :

Lançons le programme et entrons "AGE" à la question posée ; les tableaux CLE$ ET INDEX se constituent ainsi :

CLE$	
1	30
2	18
3	19
4	24
5	44
6	22
7	44

INDEX
1
2
3
4
5
6
7

Puis le système affiche :

TRI INDEXE SUR AGE

n° Enrg.	Nom	Prénom	Age
2	HUGO	VICTOR	18
3	RIMBAUD	ARTHUR	19
6	FOUCAULT	MICHEL	22
4	MAURIAC	FRANCOIS	24
1	ZOLA	EMILE	30
5	BAZIN	HERVE	44
7	LEFORT	CLAUDE	44

On voit donc que pour faire un tri, il n'est pas nécessaire d'entrer TOUTES les valeurs d'un enregistrement en mémoire mais seulement celles sur lesquelles portent le tri. Nous pouvons bien sûr conserver ces résultats sur disquette dans le cas où l'on devra les utiliser souvent. Par ailleurs, on pourra également conserver les résultats portant sur d'autres clés (on peut les voir en refaisant RUN et en entrant d'autres valeurs de clés ici).

9.3.2. L'accès indexé avec JETSAM

L'accès indexé avec Jetsam se fonde en grande partie sur des notions évoquées depuis le début de ce chapitre. Cependant, quelques nouveautés risquent de surprendre. Nous tâcherons de les rendre simples à comprendre.

Le paragraphe précédent, outre son intérêt intrinsèque, a offert l'avantage de présenter, grosso modo, ce que peut être l'accès indexé,

ses méthodes et les termes en usage. Nous allons reprendre ces notions générales dans la terminologie JETSAM en étudiant le problème suivant :

— Créer et utiliser un fichier permettant de stocker le nom, l'adresse et le numéro de téléphone de clients. Le programme devra permettre la lecture par champ du fichier, la recherche sélective, la possibilité d'effacer, de rajouter ou de modifier un enregistrement.

Appelons tout de suite les champs de l'enregistrement et leur longueur : 30 caractères pour NOM$, 30 pour ADRES$ et 12 pour TEL$.

Pour faire un fichier à accès indexé, il faut savoir que JETSAM demande en réalité la création de deux fichiers : le fichier normal de données et à accès direct et un second fichier, précisément l'index, que rant lenous nommerons dans notre exemple respectivement "CLIENT" et "INDEX". C'est bien sûr ce dernier qui va nous intéresser.

Il sera constitué de trois listes de termes dans notre exemple, faites des noms, des adresses et des téléphones de nos clients, chaque liste pouvant être considérée comme l'ensemble CLE$ + INDEX du programme illustrant le paragraphe précédent. Notez qu'un tel fichier pourra aller jusqu'à huit listes et que chaque liste porte le nom de *rang*. Il y a donc 8 rangs possibles constitués de *clés*, c'est-à-dire de mots (de 31 caractères max.), ici des noms, prénoms ou numéros de téléphone. Pour repérer les rangs, il suffira de les numéroter de 0 à 7. Dans notre exemple, l'index sera fait de trois rangs, numérotés 0, 1 et 2, et que nous pourrons appeler pour simplifier rang des noms, rang des adresses et rang des téléphones. Chaque nom (ou adresse ou tel) est une clé et un ensemble de clés identiques est appelé "jeu de clés" : si on a 123 personnes nommées "AUPATTRE" dans notre rang de noms, il y a un jeu de clés "AUPATTRE".

Important :

— Lors de la constitution de l'index, le tri effectué dans chacun des rangs est fait par JETSAM : c'est là tout l'intérêt de programmer sur l'AMSTRAD PCW 8256.

— Les rangs se suivent séquentiellement en mémoire.

Pour bien en comprendre le fonctionnement, il faut simplement imaginer que chaque rang est un fichier à lui seul. Qu'est-ce que JETSAM offre alors de plus ? de nouvelles fonctions qui vont permettre de manipuler ces rangs et ces clés.

EXEMPLE

CLIENT

1	MARTIN	3 RUE DES GLACES PARIS	11 23 32 43
2	ALBERT	12 RUE DES PRUNES LILLE	33 22 34 12
3	DUPOND	1 PLACE BLANCHE TOURS	22 34 12 89

FICHIER

INDEX

rang n°0			rang n°1			rang n°2			
1	ALBERT	2	1	1 PLACE BLANCHE TOURS	3	1	11 23 32 43	1
2	DUPOND	3	2	12 RUE DES PRUNES LILLE	2	2	22 34 12 89	3
3	MARTIN	1	3	3 RUE DES GLACES PARIS	1	3	33 22 34 12	2

9.3.2.1. La création

BUFFERS CREATE ADDREC ADDKEY FETCHREC

Créer un fichier à accès indexé demande de réserver un espace-mémoire particulier dans lequel JETSAM pourra gérer des pointeurs servant à coordonner les informations. Cette mémoire se réserve avant chaque utilisation du fichier avec la commande BUFFERS (de buffer, tampon) qui spécifie le nombre de tampons de 128 octets chacun et le manuel propose d'en mettre au moins 6. Puis il faut créer le fichier avec la commande CREATE suivie du numéro de la mémoire-tampon normale (comme en accès direct) ET du nom du fichier "CLIENT" suivi du nom de l'index "INDEX". Il faut spécifier également le verrou dont nous parlerons ultérieurement et la longueur de l'enregistrement. Les champs se réservent normalement avec FIELD.

IMPORTANT : il est obligatoire de préciser la longueur augmentée de DEUX caractères, car JETSAM se les réserve pour sa gestion.

Nous pouvons donc écrire (le programme complet sera donné en fin de chapitre) en se souvenant de l'exemple : 30 + 30 + 12 = 72 caractères de longueur :

```
REM CREATION
BUFFERS 6
CREATE #1,"CLIENT","INDEX",1,74
FIELD #1, 30 AS NOM$, 30 AS ADRES$,12 AS TEL$
```

Pour entrer les valeurs, le nom, l'adresse et le téléphone, on utilise au début exactement comme en accès direct, en passant par la mémoire-tampon :

```
INPUT "entrez le nom du client.....";N$
INPUT "entrez l'adresse.....";A$
INPUT "entrez le téléphone.....";T$
LSET NOM$=UPPER$(N$)
LSET ADRES$=UPPER$(A$)
LSET TEL$=UPPER$(T$)
```

Pour écrire le tampon sur le disque, on n'utilise plus PUT mais deux fonctions JETSAM. La première sert à la fois à écrire l'enregistrement dans le fichier de données "CLIENT" et à écrire la clé, c'est-à-dire le nom du client dans le rang des noms : cette fonction se nomme ADDREC (de add record, ajout d'un enregistrement).

Ouvrons une parenthèse ; nous parlons ici de fonctions JETSAM. Précisons qu'une fonction en général s'utilise en renvoyant une valeur codifiée. Ainsi, on ne peut utiliser une fonction telle quelle, il faut lui adjoindre une variable quelconque qui prendra la valeur codifiée. Ainsi, pour écrire avec ADDREC, nous dirons :

```
POSITION =  ADDREC ( 1, 0, 0, NOM$)
```

Cette variable POSITION aurait pu s'appeler autrement, elle n'est pas un terme JETSAM. Par contre, les paramètres suivant ADDREC sont obligatoires : ils précisent le numéro du fichier, 1, le numéro du verrou, 0, le numéro du rang des noms, 0, et la valeur de la clé à entrer dans le rang des noms, NOM$. La variable POSITION peut alors prendre plusieurs valeurs : si elle devient 0, cela signifie que tout a bien fonctionné et que l'enregistrement est entré dans le fichier "CLIENT" et que la clé du nom est écrite dans le rang des noms. Si POSITION est égal à 116, 130 ou 131, cela signifie que l'écriture a été rendue impossible parce que la fonction RANKSPEC (que nous verrons plus tard) ou qu'un verrou ont été appliqués. En d'autres termes, on peut utiliser les codes renvoyés, 0, 116, 130 ou 131, pour programmer éventuellement une gestion des erreurs.

Comme nous avons décidé que notre programme pourra lire chaque champ de l'enregistrement, nous voulons donc créer trois rangs. Ici, nous avons créé le premier, celui des noms, avec une clé que le manuel appelle une clé simple. Pour créer les autres, il ne faut pas utiliser ADDREC (qui créerait en fait d'autres fichiers) mais ADDKEY (de add key, ajout d'une clé) et une autre fonction, FETCHREC (de

fetch rec, aller chercher l'enregistrement : en fait son numéro). ADDKEY se fonde en fait pour fonctionner sur le numéro de l'enregistrement du fichier "CLIENT" qui est donné par la fonction FETCHREC, qui s'utilise suivie du numéro de fichier :

```
POSITION = ADDKEY ( 1,0,1,ADRES$,FETCHREC(1))
POSITION = ADDKEY ( 1,0,2,TEL$,FETCHREC(1))
```

On voit que d'une part on peut appeler la fonction par le même nom, POSITION, sans que cela gêne (forcément puisque cette fonction change de valeur à chaque fois) et que d'autre part les paramètres suivants ADDKEY renvoient toujours au numéro de fichier, au verrou, à la clé et au numéro de l'enregistrement.

Puis il faut maintenant refermer le fichier si on a fini d'entrer des données avec la commande CLOSE.

IMPORTANT : il ne faut pas refermer "CLIENT" et "INDEX" avec CLOSE sans indication mais avec CLOSE #1 sous peine de voir le message "inconsistent files" (fichiers incohérents) qui rendrait les fichiers impossibles à utiliser ultérieurement.

9.3.2.2. La lecture

OPEN "K" SEEKRANK SEEKNEXT SEEKPREV SEEKSET

La lecture des informations se fait en réutilisant de nouveau BUFFERS et en ouvrant le fichier "CLIENT" et "INDEX", en utilisant OPEN "K". Si on connait déjà OPEN, c'est la première fois que l'on se sert de "K". Cette lettre signifie simplement "KEY", la clé :

```
BUFFERS 6
OPEN "K", #1,"CLIENT","INDEX",2
```

notez le verrou final)

Pour lire le fichier, on utilisera en fait l'index (sinon ce n'est pas la peine de continuer !) et cet index est constitué de trois rangs, celui des noms, des adresses ou des téléphones ; il faut donc préciser lequel nous intéresse avec la fonction SEEKRANK (de seek rank, recherche du rang). Nous devons donc placer la position active (c'est-à-dire le pointeur de recherche) sur la première clé du rang désiré puis faire une lecture séquentielle du rang jusqu'à la fin en lisant chaque clé :

```
POSITION = SEEKRANK(1,0,0)
    WHILE POSITION = 0 OR POSITION = 101
        GET #1
```

```
    PRINT NOM$,ADRES$,TEL$
    POSITION= SEEKNEXT(1,0)
WEND
```

SEEKRANK a trois paramètres : fichier, verrou et rang.

On fait une boucle avec while et wend qui teste si la position actuelle est sur une clé du rang (0) ou sur un jeu de clés (101) suivant : 0 et 101 sont deux valeurs positives de la fonction. Si oui, alors on va chercher l'enregistrement avec GET, sans qu'il soit besoin de préciser le numéro (on peut le savoir d'ailleurs avec FETCHREC(1)), puis on passe au suivant avec SEEKNEXT (de seek next, chercher le suivant), qui a deux paramètres, fichier et verrou. Nous pouvons améliorer la lecture du rang en utilisant à la place de SEEKNEXT la fonction SEEKSET (recherche de l'ensemble). Par exemple, nous pourrions imaginer lire un index de livres et chercher tous les auteurs différents. Si notre fichier de livres comporte 45 ouvrages de FREUD, le programme précédent affichera 45 fois FREUD ; par contre, si on utilise SEEKSET à la place de SEEKNEXT, le programme lira la première clé du jeu de clés "FREUD" et sautera directement à la clé située juste après le 45ème "FREUD" du fichier. Notez également qu'il est possible de lire l'index non pas du début vers la fin mais de la fin vers le début avec SEEKPREV (de seek previous, recherche du précédent) : les fonctions SEEKSET et SEEKPREV s'utilisent exactement comme SEEKNEXT.

9.3.2.3. L'ajout d'enregistrements et de clés

CONSOLIDATE

Il se fait simplement en ouvrant le fichier comme précédemment et en entrant les données en précisant la fonction CONSOLIDATE. Il ne faut pas oublier que le fait de rajouter des informations à un fichier qui en possède déjà oblige JETSAM à modifier les valeurs de clés triées ; il doit donc gérer tout cela dans ses buffers et coordonner l'index "INDEX " avec le fichier "CLIENT". Oublier de consolider l'ensemble c'est s'attendre à rendre incohérents entre eux les pointeurs. Il ne faut pas oublier que JETSAM doit mettre à jour en permanence ses informations et les traite au fur et à mesure de leur entrée dans ses BUFFERS.

```
BUFFERS 6
OPEN "K",#1,"CLIENT","INDEX",2
FIELD #1,30 AS NOM$,30 AS ADRES$,12 AS TEL$
    .... ENTREE DE DONNEES COMME EN CREATION
FERMETURE = CONSOLIDATE(1)
CLOSE #1
```

9.3.2.4. La recherche de données

SEEKKEY

La recherche de données se fait comme la lecture à ceci près que l'on peut utiliser une autre fonction, SEEKKEY (de seek key, recherche de la clé). Ainsi, si on cherche à partir du nom de quelqu'un son adresse et son téléphone :

```
BUFFERS 6
OPEN "K", #1,"CLIENT","INDEX",2
FIELD #1, 30 AS NOM$, 30 AS ADRES$, 12 AS TEL$
INPUT "entrez le nom du client ";NR$
NR$=LEFT$(NR$,30):NR$=UPPER$(NR$+ STRING$(30 - LEN(NR$)," ")
POSITION = SEEKKEY (1,0,0,NR$)
    WHILE POSITION = 0
    GET #1
    PRINT NOM$,ADRES$,TEL$
    POSITION = SEEKNEXT ( 1,0)
    WEND
 CLOSE #1
```

On notera la ligne suivant l'entrée du nom : elle sert à la fois à raccourcir le nom si éventuellement il dépasse 30 caractères et à rajouter des espaces s'il est plus petit que 30.

9.3.2.5. L'effacement de données

DELKEY

Pour effacer un enregistrement, il faut l'effacer du fichier de données, "CLIENT" dans notre exemple. Mais il faut également effacer toutes les clés correspondantes dans les rangs existants pour remettre à jour le fichier "INDEX". Il faut donc entrer le client à effacer, faire une recherche dans le rang des noms (si on veut effacer à partir du nom) et utiliser la fonction DELKEY (de delete key, effacer la clé). Le principe est simple : si il y a un seul rang, effacer la clé dans ce rang efface en même temps l'enregistrement correspondant dans le fichier "CLIENT". En revanche, s'il existe plusieurs rangs l'enregistrement n'est pas effacé tant que la clé du dernier rang ne l'est pas : heureusement d'ailleurs sinon on ne pourrait pas modifier le fichier. La fonction DELKEY s'utilise avec les paramètres habituels.

```
BUFFERS 6
OPEN "K", #1,"CLIENT","INDEX",2
FIELD #1, 30 AS NOM$,30 AS ADRES$,12 AS TEL$
INPUT "entrez le client à effacer ";NR$
NR$=LEFT$(NR$,30):NR$=UPPER$(NR$+ STRING$(30 - LEN(NR$)," ")
POSITION = SEEKKEY (1,0,0,NR$)
```

```
    WHILE POSITION = 0
        GET #1
        F=FETCHREC(1)
        POSITION = DELKEY ( 1,0,0,NOM$,F)
        POSITION = DELKEY ( 1,0,1,ADRES$,F)
        POSITION = DELKEY ( 1,0,2,TEL$,F)
    POSITION = SEEKNEXT ( 1,0)
    WEND
    FERMETURE = CONSOLIDATE(1)
CLOSE #1
```

On voit que, grâce à FETCHREC, JETSAM peut effacer les clés correspondant au numéro de l'enregistrement.

9.3.2.6. La modification de données

PUT

Imaginons que nous voulons pouvoir modifier les numéros de téléphone. Il faut donc pour cela modifier une partie de l'enregistrement de "CLIENT" et changer la clé correspondante dans le rang n°2. Nous pouvons modifier d'une manière brutale et peu élégante en effaçant l'enregistrement et en ajoutant le nouvel enregistrement. Bien que possible, cette façon de faire oblige à réécrire des renseignements bons (le nom et l'adresse) ; mieux vaut modifier directement.

Nous savons que pour modifier il faut d'abord savoir où se trouve l'enregistrement à modifier, connaître son numéro dans "CLIENT" et l'entrer dans la mémoire-tampon, avec GET ; puis il faut entrer dans le champ téléphone, soit TEL$, le nouveau numéro téléphonique et réécrire l'enregistrement sur le disque avec PUT (eh oui, ici il faut utiliser PUT et non ADDREC ou ADDKEY, car ces dernières fonctions rajouteraient un nouvel enregistrement au fichier). Or, le numéro de téléphone étant une clé du rang n°2, il faut réécrire la bonne clé, la nouvelle, à la place de l'ancienne. Pour cela, on ajoute la bonne clé avec ADDKEY et on efface l'ancienne avec DELKEY.

IMPORTANT : on sait que DELKEY efface la clé ET l'enregistrement correspondant s'il n'existe qu'un seul rang ; il faut donc AJOUTER en premier et EFFACER en second les clés en question.

```
BUFFERS 6
OPEN "K", #1,"CLIENT","INDEX",2
FIELD #1, 30 AS NOM$,30 AS ADRES$,12 AS TEL$
INPUT "entrez le numéro de téléphone à modifier ";NC$
INPUT "entrez le nouveau numéro";NT$
    NC$=LEFT$(NC$,12):NC$=UPPER$(NC$+ STRING$(12 - LEN(NC$)," ")
    POSITION = SEEKKEY (1,0,2,NC$)
    WHILE POSITION = 0
```

```
        GET #1
        LSET TEL$ = NT$:PUT #1:POSITION=ADDKEY( 1,0,2,NT$,
        FETCHREC(1)):POSITION= DELKEY(1,0,2,NC$,FETCHREC(1)):
        POSITION=SEEKNEXT(1,0)
    WEND
FERMETURE =CONSOLIDATE(1)
CLOSE #1
```

Voici, en guise de conclusion sur ce paragraphe, le programme complet et amélioré sur l'exemple traité :

```
10 REM
20 PRINT "1 CREATION 2 LECTURE 3 AJOUT  4 RECHERCHE  5 EFFACEMENT 6
MODIFICATION
30 INPUT "Entrez votre  choix ";c
40 ON C GOSUB  70,160,360,440,660,1010
50 GOTO 20
60 '
70 REM CREATION
80 '
90 BUFFERS 6
100 CREATE #1,"CLIENT","INDEX",1,74
110 FIELD #1,30 AS NOM$,30 AS ADRES$,12 AS TEL$
120 GOSUB 860
130 CLOSE #1
140 RETURN
150 '
160 REM LECTURE
170 '
180 BUFFERS 6
190 OPEN "K",#1,"CLIENT","INDEX",2
200 FIELD #1,30 AS NOM$,30 AS ADRES$,12 AS TEL$
210 PRINT "0  NOM    1  ADRESSE  2  TELEPHONE
220 PRINT "  9  RETOUR MENU PRINCIPAL   9
230 INPUT "entrez votre choix ";c
240 IF C=9 THEN CLOSE #1:RETURN
250 POSITION=SEEKRANK(1,0,C)
260      WHILE POSITION = 0   OR POSITION =101
270         GET #1
280         PRINT NOM$;TAB(32);ADRES$;TAB(66);TEL$
290         POSITION=SEEKNEXT(1,0)
300      WEND
310 PRINT "fin de la lecture..............."
320 GOTO 210
330 '
340 REM AJOUT D ENREGISTREMENTS
350 '
360 BUFFERS 6
370 OPEN "K",#1,"CLIENT","INDEX",2
380 FIELD #1,30 AS NOM$,30 AS ADRES$,12 AS TEL$
390 GOSUB 860
400 FERMETURE=CONSOLIDATE(1)
410 CLOSE #1
420 RETURN
430 '
440 REM RECHERCHE
450 '
460 BUFFERS 6
470 OPEN "K",#1,"CLIENT","INDEX",2
```

```
480 FIELD #1,30 AS NOM$,30 AS ADRES$,12 AS TEL$
490 PRINT "---------------- RECHERCHES DIVERSES ------------------
500 PRINT "0 NOM  1 ADRESSE  2 TELEPHONE
510 PRINT "9 RETOUR MENU PRINCIPAL "
520 INPUT "Entrez votre choix ";C
530 IF c=9 THEN CLOSE #1:RETURN
540 INPUT "Entrez le   MOT / TERME recherché ";nr$
550   IF C=0 OR C=1 THEN NR$=LEFT$(NR$,30):NR$=UPPER$(
NR$+STRING$(30-LEN(NR$)," "))
560  IF C=2 THEN NR$=LEFT$(NR$,12):NR$=UPPER$( NR$+STRING$(12-LEN(NR$),"
"))
570       POSITION=SEEKKEY(1,0,C,NR$)
580           WHILE  POSITION = 0
590             GET #1
600        PRINT NOM$;TAB(32);ADRES$;TAB(66);TEL$
610             POSITION=SEEKNEXT(1,0)
620           WEND
630 PRINT "Fin des recherches. "
640 GOTO 500
650 '
660 REM EFFACEMENT D UN ENREGISTREMENT
670 '
680 OPEN "K",#1,"CLIENT","INDEX",2
690 FIELD #1,30 AS NOM$,30 AS ADRES$,12 AS TEL$
700 INPUT "ENTREZ LE NOM A EFFACER ";NR$
710    NR$=UPPER$(NR$+  STRING$( 30 -LEN(NR$), " "))
720 POSITION=SEEKKEY(1,0,0,NR$)
730    WHILE POSITION=0
740           GET #1
750           F=FETCHREC(1)
760           POSITION=DELKEY(1,0,0,NOM$,F)
770           POSITION=DELKEY(1,0,1,ADRES$,F)
780           POSITION=DELKEY(1,0,2,TEL$,F)
790           GOTO 820
800      POSITION=SEEKNEXT(1,0)
810   WEND
820 FERMETURE=CONSOLIDATE(1)
830 CLOSE #1
840 RETURN
850 '
860 REM SOUS-PROGRAMME D ' ENTREE DES ENREGISTREMENTS
870 '
880 '
890 INPUT " Entrez le nom du client..... ";N$
900 INPUT " Entrez l'adresse............. ";A$
910 INPUT " Entrez le téléphone.......... ";T$
920    LSET NOM$=UPPER$(N$)
930    LSET ADRES$=UPPER$(A$)
940    LSET TEL$=UPPER$(T$)
950 POSITION=ADDREC(1,0,0,NOM$)
960 POSITION=ADDKEY(1,0,1,ADRES$,FETCHREC(1))
970 POSITION=ADDKEY(1,0,2,TEL$,FETCHREC(1))
980 INPUT " On continue  OUI / RETURN  NON / TOUCHE QUELCONQUE ";ch$:IF
ch$="" THEN 890
990 RETURN
1000 '
1010 REM MODIFICATION
1020 '
1030 BUFFERS 6
1040 OPEN "K",#1,"CLIENT","INDEX",0
1050 FIELD #1,30 AS NOM$,30 AS ADRES$,12 AS TEL$
```

```
1060 INPUT "Entrez le  numéro de téléphone du client à modifier ";NC$
1070 INPUT "entrez le nouveau numéro de téléphone";NT$
1080   NC$=LEFT$(NC$,12):NC$=UPPER$( NC$+STRING$(12-LEN(NC$)," "))
1090   NT$=LEFT$(NT$,12):NT$=UPPER$( NT$+STRING$(12-LEN(NT$)," "))
1100      POSITION=SEEKKEY(1,0,2,NC$)
1110       WHILE POSITION=0 OR POSITION=101
1120         GET #1
1130         LSET TEL$=NT$:PUT #1:POSITION=ADDKEY(1,0,2,NT$,FETCHREC(1
)):POSITION=DELKEY(1,0,2,NC$,FETCHREC(1)):GOTO 1160
1140         POSITION=SEEKNEXT(1,0)
1150       WEND
1160     FERMETURE=CONSOLIDATE(1)
1170     CLOSE #1
1180 RETURN
```

9.3.3. Les nombres et leur conversion

Pour utiliser des fichiers, nous savons qu'il faut convertir les valeurs numériques en valeurs alphanumériques parce que l'on ne peut écrire que des caractères sur les disques. Tout ce que nous avons vu avec MKI$, MKS$, et MKD$ reste valable ; cependant, en ce qui concerne les index, il faut utiliser d'autres fonctions.

DEC$, MKIK$, MKUK$, CVIK, CVUK

Pour écrire des valeurs numériques en tant que clés, il faut utiliser la fonction DEC$, que l'on peut tester directement ainsi :

```
A$ = DEC$( 12345 , "#####.##")
PRINT A$
12345.00
```

On voit qu'il faut utiliser DEC$ comme nous le faisons avec USING, en donnant un format à notre valeur numérique, ce qui transforme le nombre en chaîne de caractères.

Pour utiliser cette fonction dans un fichier, on peut prendre l'exemple d'une valeur en francs à stocker dans un rang donné. Comme cette valeur servira de clé, il faut user de DEC$ ainsi :

```
INPUT "valeur en francs .....";  FRANC
LSET PRIX$= DEC$(FRANC , "#######.##")
```

Ici, on utilise un champ quelconque nommé PRIX$ qui a 10 caractères, 9 dièses et un point décimal ; si on entre à la question la valeur 123. 34, on aura :

PRIX$ = " | | | | | 1 | 2 | 3 | . | 3 | 4 |

Bien sûr, si on recherche ensuite une telle valeur dans un rang, il faut entrer 123.34 dans une chaîne ET ajouter les espaces manquants au DEBUT de cette chaîne :

```
INPUT ''entrez la valeur cherchée'' ;NR$
NR$ = STRING$(10 - LEN(NR$), '' '') + NR$
```

IMPORTANT : il faut prendre en compte le signe '' – '' avec les nombres négatifs. La lecture de nombres positifs et négatifs peut sembler curieuse ; par exemple la série suivante est triée, malgré l'évidence :

```
 - 1
- 10
 100
 123
- 12000
```

En fait, les espaces précédents les nombres sont pris en compte par le tri de l'index. Il faut donc opérer autrement. Un procédé simple est d'utiliser deux rangs de valeurs : un positif et un négatif.

Il ne faut pas oublier également le signe '' – '' dans le comptage des dièses, car il prend bien sûr un espace dans le format.

Si on désire entrer des valeurs entières, on peut utiliser MKIK$ ou MKUK$.

Dans l'intervalle – 32768 à + 32767 : MKIK$

Dans l'intervalle 0 à + 65535 et pour des valeurs sans signe : MKUK$

Leur conversion se fait évidemment avec CVIK et CVUK.

Il faut rappeler que ces trois fonctions sont les seules qui permettent d'écrire des nombres dans un index.

9.3.4. Diverses fonctions JETSAM

DISPLAY FETCHKEY$ FETCHRANK FETCHREC RANKSPEC LOCK

On peut rapidement visualiser, bien que cela ne soit pas très clair, un fichier en entrant :

```
DISPLAY ''nom du fichier''. (de display, voir sur l'écran).
```

On a bien compris qu'une valeur est essentielle pour la bonne gestion des fichiers, c'est la position de la clé active, c'est-à-dire là où se trouve le pointeur. Cette position se réfère à une clé donc, avec FETCHKEY$ (de fetch key, recherche de la clé) :

PRINT FETCHKEY$(1)

où (1) représente le numéro de fichier.

Le numéro de l'enregistrement correspondant à la clé actuellement en usage est donné avec FETCHREC (de fetch rec, recherche de l'enregistrement) :

PRINT FETCHREC (1)

Et le rang dans lequel se trouve la clé active est donné par FETCHRANK (de fetch rank, recherche du rang).

PRINT FETCHRANK (1)

IMPORTANT : on ne peut être sûr totalement que les valeurs renvoyées soient bonnes, car ces fonctions font référence à la dernière valeur utilisée : elles pourraient être en fait déjà effacées autrement sans que le système ne le sache, à moins d'avoir verrouillé le fichier ou l'enregistrement.

• Le verrou

Un fichier ou un enregistrement peuvent être verrouillés en lecture, en écriture, ou déverrouillés. Il y a donc trois codes correspondants, respectivement 1, 2, 0.

Le verrouillage a une fonction essentielle en utilisation multiple. Si plusieurs personnes sont branchées sur le même fichier, il faut régler les modalités d'accès aux données de telle manière qu'un utilisateur n'ait pas des informations erronées, ou en cours de modification. Le verrouillage n'intéresse pas à priori la personne utilisant seule son ordinateur, sauf si elle se sert de la fonction RANKSPEC (cf ci-dessous).

On a remarqué qu'un verrou s'applique dans une déclaration d'ouverture de fichier ou dans les fonctions JETSAM. On peut verrouiller avec la commande LOCK (de lock, verrouiller) :

VERROU = LOCK (1,0,12)

Ici, on applique un deverrouillage (0) dans le fichier n°1 au 12ème enregistrement.

• Particularité

On peut avoir besoin pour diverses raisons de ne pas avoir des ''doublons'' dans un index, c'est-à-dire des clés identiques. On empêchera le système d'écrire ces doublons avec RANKSPEC (de rank spécification : précision dans le rang) :

LIEU = RANKSPEC (1,1,1)

Ici, le premier ''1'' spécifie le fichier, le second ''1'' le rang et le troisième le verrou : contrairement aux autres fonctions, le verrou est bien en dernière position et toujours égal à 1.

Exercice

Reprendre le programme donné en exemple précédemment. Il permettra de gérer un fichier ''CLIENT'', qui fera apparaître, outre le nom, l'adresse et le téléphone, un compte qui pourra être un débit ou un crédit. Le programme devra être simplifié et permettra de plus d'effacer à la demande le fichier.

Solution

```
10 REM
20 PRINT "1 CREATION 2 LECTURE 3 AJOUT  4 RECHERCHE  5 EFFACEMENT 6
MODIFICATION 7 EFFACEMENT TOTAL
30 INPUT "Entrez votre  choix ";c
40 ON C GOSUB  70,160,410,490,700,1100,1400
50 GOTO 20
60 '
70 REM------------- CREATION------------------------------
80 '
90 BUFFERS 10
100 CREATE #1,"CLIENT","INDEX",1,85
110 FIELD #1,30 AS NOM$,30 AS ADRES$,12 AS TEL$,11 AS MON$
120 GOSUB 910
130 CLOSE #1
140 RETURN
150 '
160 REM ------------LECTURE------------------------------
170 '
180 GOSUB 1300
190 PRINT:PRINT:PRINT
200 INPUT "0 ECRAN   ----   1 IMPRIMANTE ";P
210 PRINT "0  NOM    1  ADRESSE  2  TELEPHONE 3 CREDIT 4 DEBIT
220 PRINT "  9  RETOUR MENU PRINCIPAL   9
230 INPUT "entrez votre choix ";c
240 IF C=9 THEN CLOSE #1:RETURN
250 PRINT:PRINT:PRINT
260 TOTPOS =0:TOTNEG=0
270 POSITION=SEEKRANK(1,0,C)
280      WHILE POSITION = 0   OR POSITION =101
290         GET #1
300 GOSUB 1370
310 VA= VAL (MON$)
320 IF VA < 0 THEN TOTNEG=TOTNEG + VA ELSE TOTPOS=TOTPOS+VA
330         POSITION=SEEKNEXT(1,0)
340      WEND
350 IF P=0 THEN PRINT:PRINT "total débits = ";TOTNEG,"total crédits
=";TOTPOS
360 IF P=1 THEN LPRINT:LPRINT "total débits = ";TOTNEG,"total crédits
=";TOTPOS
370 PRINT
380 PRINT "fin de la lecture....."
390 GOTO 210
400 '
410 REM ----------------AJOUT D ENREGISTREMENTS----------------
420 '
430 GOSUB 1300
440 GOSUB 910
450 FERMETURE=CONSOLIDATE(1)
```

```
460 CLOSE #1
470 RETURN
480 '
490 REM --------------------RECHERCHE-----------------------------
500 '
510 GOSUB 1300
520 PRINT "---------------- RECHERCHES DIVERSES ------------------
530 PRINT "0 NOM  1 ADRESSE  2  TELEPHONE 3  CREDIT 4  DEBIT
540 PRINT "9 RETOUR MENU PRINCIPAL "
550 INPUT "Entrez votre choix ";C
560 IF c=9 THEN CLOSE #1:RETURN
570 INPUT "Entrez le   MOT / TERME recherch( ";nr$
580   IF C=0 OR C=1 THEN NR$=LEFT$(NR$,30):NR$=UPPER$(
NR$+STRING$(30-LEN(NR$)," "))
590  IF C=2 THEN NR$=LEFT$(NR$,12):NR$=UPPER$( NR$+STRING$(12-LEN(NR$),"
"))
600 IF C=3 OR C=4  THEN NR$=LEFT$(NR$,11):NR$= STRING$(11-LEN(NR$)," ")+
NR$
610       POSITION=SEEKKEY(1,0,C,NR$)
620           WHILE  POSITION = 0
630             GET #1
640 GOSUB 1370
650             POSITION=SEEKNEXT(1,0)
660           WEND
670 PRINT "Fin des recherches. "
680 GOTO 530
690 '
700 REM ---------- EFFACEMENT D UN ENREGISTREMENT------------------
710 '
720 GOSUB 1300
730 INPUT "entrez le nom du client @ effacer du fichier";NR$
740    NR$=UPPER$(NR$+  STRING$( 30 -LEN(NR$), " "))
750 POSITION=SEEKKEY(1,0,0,NR$)
760    WHILE POSITION=0
770           GET #1
780           F=FETCHREC(1)
790           POSITION=DELKEY(1,0,0,NOM$,F)
800           POSITION=DELKEY(1,0,1,ADRES$,F)
810           POSITION=DELKEY(1,0,2,TEL$,F)
820           POSITION=DELKEY(1,0,3,MON$,F)
830           POSITION=DELKEY(1,0,4,MON$,F)
840           GOTO 870
850      POSITION=SEEKNEXT(1,0)
860   WEND
870 FERMETURE=CONSOLIDATE(1)
880 CLOSE #1
890 RETURN
900 '
910 REM ----SOUS-PROGRAMME D ' ENTREE DES ENREGISTREMENTS---------
920 '
930 '
940 INPUT " Entrez le nom du client..... ";N$
950 INPUT " Entrez l'adresse..... ";A$
960 INPUT " Entrez le t(l(phone..... ";T$
970 INPUT "entrez le montant..... ";M
980   LSET NOM$=UPPER$(N$)
990   LSET ADRES$=UPPER$(A$)
1000   LSET TEL$=UPPER$(T$)
1010   LSET MON$=DEC$(M,"########.##")
1020 POSITION=ADDREC(1,0,0,NOM$)
1030 POSITION=ADDKEY(1,0,1,ADRES$,FETCHREC(1))
```

```
1040 POSITION=ADDKEY(1,0,2,TEL$,FETCHREC(1))
1050 IF M >= 0 THEN POSITION=ADDKEY(1,0,3,MON$,FETCHREC(1))
1060 IF M < 0  THEN POSITION=ADDKEY(1,0,4,MON$,FETCHREC(1))
1070 INPUT " On continue  OUI / RETURN  NON / TOUCHE QUELCONQUE ";ch$:IF
ch$="" THEN 940
1080 RETURN
1090 '
1100 REM -------------MODIFICATION-----------------------
1110 '
1120 GOSUB 1300
1130 INPUT "Entrez le  num{ro de t{l{phone du client @ modifier ";NC$
1140 INPUT "entrez le nouveau num{ro de t{l{phone";NT$
1150   NC$=LEFT$(NC$,12):NC$=UPPER$( NC$+STRING$(12-LEN(NC$)," "))
1160  NT$=LEFT$(NT$,12):NT$=UPPER$( NT$+STRING$(12-LEN(NT$)," "))
1170       POSITION=SEEKKEY(1,0,2,NC$)
1180        WHILE POSITION=0 OR POSITION=101
1190          GET #1
1200          LSET TEL$=NT$:PUT #1:POSITION=ADDKEY(1,0,2,NT$,FETCHREC(1 ))
:POSITION=DELKEY(1,0,2,NC$,FETCHREC(1)):GOTO 1240
1210          POSITION=SEEKNEXT(1,0)
1220        WEND
1230 REM
1240      FERMETURE=CONSOLIDATE(1)
1250      CLOSE #1
1260 RETURN
1270 REM
1280 REM ------DECLARATION DES CHAMPS----------------------
1290 REM
1300 BUFFERS 10
1310 OPEN "K",#1,"CLIENT","INDEX",0
1320 FIELD #1,30 AS NOM$,30 AS ADRES$,12 AS TEL$,11 AS MON$
1330 RETURN
1340 REM
1350 REM ------AFFICHAGE DES VALEURS----------------------
1360 REM
1370 IF P=0 THEN PRINT "* ";
NOM$;TAB(33);ADRES$;TAB(65);TEL$;TAB(78)"*";MON$
1380 IF P=1 THEN LPRINT CHR$(27)+CHR$(15):LPRINT"* ";
NOM$;TAB(33);ADRES$;TAB(65);TEL$;TAB(78)"*";MON$
1390 RETURN
1400 REM
1410 REM -------EFFACEMENT DU FICHIER----------------------
1420 INPUT "veuillez confirmer O/N ";C$
1430 IF c$="o" OR c$="O" THEN KILL "client":KILL"index"
1440 RETURN
```

Commentaires :

De 1280 à 1340 : déclaration des champs ; on en a réservé un au compte, MON$.

De 1350 à 1390 : test si on désire l'affichage sur l'écran ou l'imprimante.

De 1410 à 1440 : on efface le fichier après demande de confirmation : c'est préférable ! On note qu'il faut effacer le fichier "CLIENT" et l'index.

La partie intéressante commence à la ligne 1010. Ici, on utilise DEC$ pour formater une chaine de 11 caractères. Puis en lignes 1050 et 1060, un test est effectué sur la qualité du compte : selon qu'il est positif ou négatif, il sera entré dans le rang n°3 ou le rang n°4 ; on a donc créé DEUX rangs spécialisés pour le compte, ce qui résout le problème des espaces et du formatage des nombres négatifs. On retrouvera ces rangs à la lecture ou à la modification.

Un aspect intéressant également en ligne 260 et suivantes. On peut à la lecture vouloir connaître le montant des débits et celui des crédits ; c'est avec TOTPOS et TOTNEG, pour total positif et total négatif.

Voici un exemple d'utilisation :

CRÉATION :

```
*ALBERT   34 PLACE DE LA POSTE TOURS          45 54 43 344  *     45.66
*DUPONT   12 RUE DES ALOUETTES PARIS          12 23 43 12   *  12434.45
*DUPONT   45 RUE DES CERISES SEDAN            45 65 22 11   *  -4567.88
*DURAND   1 ALLEE DES PRUNES SEDAN            23 43 76 87   *  -9030.44
 total débits = -13598.32     total crédits = 12480.11
```

(lecture faite après création sur le rang des noms)

LECTURES DIVERSES SUR CREDITS ET DEBITS

```
*ALBERT   34 PLACE DE LA POSTE TOURS          45 54 43 344  *     45.66
*DUPONT   12 RUE DES ALOUETTES PARIS          12 23 43 12   *  12434.45
 total débits = 0     total crédits = 12480.11
```

```
*DUPONT   45 RUE DES CERISES SEDAN            45 65 22 11   *  -4567.88
*DURAND   1 ALLEE DES PRUNES SEDAN            23 43 76 87   *  -9030.44
 total débits = -13598.32     total crédits =0
```

AJOUT

```
*ALBERT   34 PLACE DE LA POSTE TOURS          45 54 43 344  *    45.66
*BIHAN    12 RUE DES ORDINATEURS SEDAN        01 01 01 11   *  6990.00
*DUPONT   12 RUE DES ALOUETTES PARIS          12 23 43 12   *  12434.45
*DUPONT   45 RUE DES CERISES SEDAN            45 65 22 11   *  -4567.88
*DURAND   1 ALLEE DES PRUNES SEDAN            23 43 76 87   *  -9030.44
 total débits = -13598.32     total crédits = 19470.11
```

(lecture sur rang des noms après ajout)

EFFACEMENT

```
*BIHAN    12 RUE DES ORDINATEURS SEDAN        01 01 01 11   *  6990.00
*DUPONT   12 RUE DES ALOUETTES PARIS          12 23 43 12   *  12434.45
*DUPONT   45 RUE DES CERISES SEDAN            45 65 22 11   *  -4567.88
*DURAND   1 ALLEE DES PRUNES SEDAN            23 43 76 87   *  -9030.44
 total débits = -13598.32     total crédits = 19424.45
```

(effacement de ALBERT et lecture sur rang des noms)

LECTURE DIVERSES DES DEBITS ET CREDITS

```
*BIHAN    12 DES ORDINATEURS SEDAN            01 01 01 11   *  6990.00
*DUPONT   12 RUE DES ALOUETTES PARIS          12 23 43 12   *  12434.45
 total débits = 0     total crédits = 19424.45
```

```
*DUPONT   45 RUE DES CERISES SEDAN            45 65 22 11   *  -4567.88
*DURAND   1 ALLEE DES PRUNES SEDAN            23 43 76 87   *  -9030.44
 total débits = -13598.32     total crédits = 0
```

Dépôt légal : juillet 1986
N° d'Editeur : 4473

Imprimerie BOUDIN - Paris

Photocomposition : COMPO SPRINT - 28240 La Loupe

Cet ouvrage est écrit selon une progression éprouvée en cours (l'auteur enseigne l'informatique principalement à des adultes).

Il est destiné à interesser le débutant puisque conçu dans un souci didactique, en supposant la programmation inconnue, et en même temps susceptible d'intéresser le programmeur confirmé avec l'important chapitre réservé aux fichiers . En effet, le langage Basic de l'AMSTRAD PCW 8256 / 8512 utilise JETSAM, le gestionnaire intégré de fichiers à accès indexé et ses nouvelles fonctions. Ainsi, l'ouvrage couvre une large part de ce qu'il est possible de faire avec le langage BASIC.

Il est divisé en neuf chapitres qui se terminent tous par une série d'exercices dont la solution est toujours commentée et expliquée.

EYROLLES

8355

couverture JÉRÔME LO MONACO.