données. Leur emploi représente un excellent moyen de contrôle lors de la saisie. Côté utilisateur, ils permettent de corriger à tout moment les données déjà tapées, ce qui s'avère impossible avec l'instruction INPUT qui provoque l'acquisition immédiate des données par l'ordinateur. L'écriture de programmes ayant recours aux masques de saisie présente incontestablement des difficultés. Nous aborderons donc cette étude progressivement et nous commencerons par l'exemple simple d'un sous-programme d'entrée et de contrôle de la date.

L'émission de la date est indispensable dans de nombreuses applications, telles que le classement d'états ou l'édition de documents comptables. Tout logiciel sérieux se doit de vérifier la cohérence de la date entrée par l'opérateur. Nous nous bornerons, dans notre exemple, à contrôler la conformité du mois et du jour entrés de façon à ce que le dernier ne soit pas supérieur au nombre de jours du mois concerné. L'organigramme de ce sous-programme se trouve page 600.

Il faut prévoir, parallèlement à cette routine, deux tables à une dimension, l'une pour les noms ou sigles des douze mois et l'autre pour les nombres de jours correspondants.

Voici comment s'effectuera alors le contrôle. Avant d'appeler le sous-programme, on aura réparti la date entrée dans trois chaînes de deux caractères : JOUR$, MOIS$ et ANNEE$. On pourra donc, désormais, comparer le contenu de la chaîne MOIS$ aux douze valeurs prévues. S'il n'est égal à aucune d'elles, c'est que le mois aura été mal tapé. Si, au contraire, on trouve MOIS$ dans la table, son numéro permettra de trouver dans la seconde table le nombre de jours qu'il contient. Le contrôle de la validité du jour consistera alors à vérifier, après l'avoir converti en numérique, qu'il est inférieur ou égal au nombre de jours du mois indiqué. Seul le mois de février fait exception. Pour ce mois, on est obligé de regarder si l'année est, ou non, bissextile. Ce contrôle est effectué dans un autre sous-programme.

Solutions du test 17

1 / Sur cette imprimante, les caractères sont formés par une juxtaposition de points parmi ceux d'une matrice déterminée. Ces points sont obtenus par l'impact d'aiguilles réunies dans une tête d'impression qui est active dans les deux sens de son déplacement (aller et retour). Une ligne est constituée de 132 caractères maximum imprimés à une vitesse de 120 par seconde. Une mémoire tampon de 4 K (4 096 octets) permet de libérer plus rapidement l'ordinateur lors des travaux d'impression.

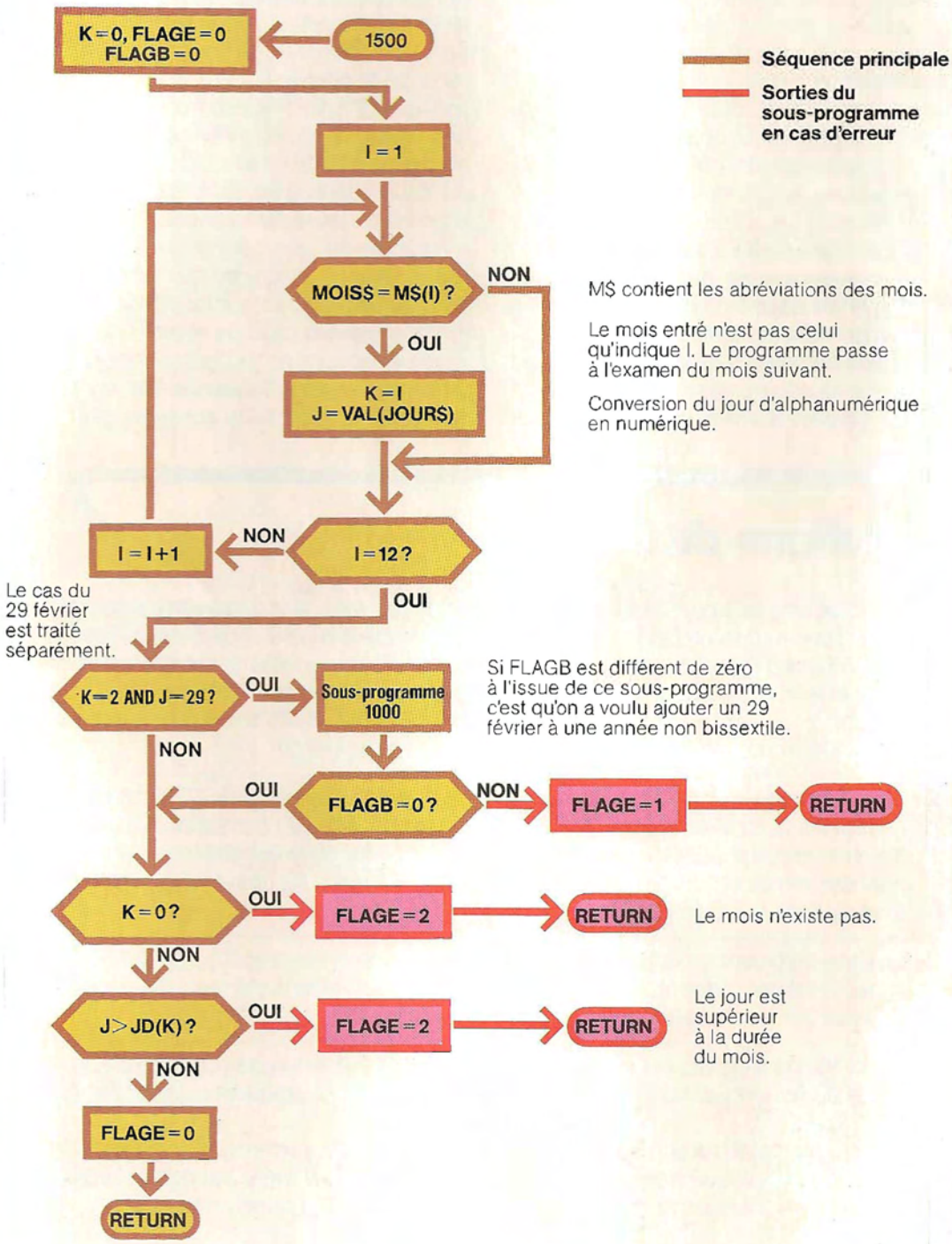
2 / 33 et 110 sont les codes respectifs des caractères ! et n. En revanche, 27 et 10 sont les codes de « caractères spéciaux » ou caractères de contrôle. 27 correspond à ESCAPE. Il avertit un périphérique (imprimante ou terminal vidéo) que les données qui suivent sont des ordres et non des caractères à imprimer. Quant à 10, c'est le code réservé au saut de ligne (LINE FEED).

3 / La ligne 20 provoque l'affichage de la chaîne A$ et de la variable C. L'instruction de la ligne 30 est par contre incorrecte car elle emploie pour l'édition d'une chaîne de caractères un format réservé aux valeurs numériques.

4 / Ligne 10 : les cinq valeurs sont imprimées au début de cinq lignes consécutives.
Ligne 20 : les cinq valeurs sont imprimées à la suite les unes des autres sans séparation (symbole ;).
Ligne 30 : les cinq valeurs seront imprimées à partir de la dixième colonne. Cependant, le ; après A(I) empêche le retour du chariot (passage à la ligne suivante). Les cinq valeurs seraient donc superposées si l'imprimante pouvait revenir en arrière.

SOUS-PROGRAMME DE CONTROLE MOIS-JOUR

Entrées : JOUR$ = chaîne contenant le jour (nombres de 1 à 31) ;
MOIS$ = chaîne contenant le mois (abréviation) ;
ANNEE$ = chaîne contenant l'année (pour les années bissextiles).
Sortie : FLAGE > 0 en cas d'erreur, FLAGE = 0 pour une date correcte.



On y vérifie que l'année est divisible par quatre (voir schéma ci-dessous). N'oublions pas, cependant, que les années séculaires ne sont bissextiles que si elles sont divisibles par quatre cents (1900 ne l'était pas, 2000 le sera). On a prévu, dans ces sous-programmes, plusieurs flags qui indiqueront le type d'erreur éventuellement commis.
Les pages 602 et 603 retracent l'organigramme d'un sous-programme de lecture et de contrôle d'une date entrée au clavier, qui réunit les différentes routines que nous venons de présenter, et illustre un exemple d'acquisition des données à l'aide d'un masque de saisie. L'instruction INPUT constitue le moyen le plus simple et le plus direct de communiquer au programme les données en entrée. Ainsi, l'instruction

INPUT JOUR$, MOIS$, ANNEE$

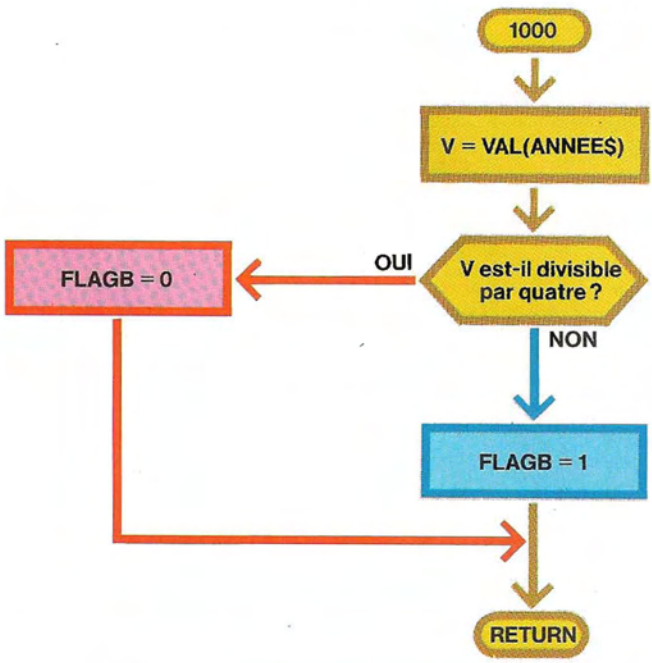
permet l'acquisition des trois éléments (chaînes de caractères) de la date.
Toutefois, ce mode de saisie présente deux inconvénients majeurs. Tout d'abord, rien n'indique à l'utilisateur sous quelle forme il doit taper les données. Il ignorera donc, par exemple, combien de caractères représentent le mois (deux caractères numériques, ou jusqu'à neuf caractères alphabétiques). De plus, la ligne d'affichage de la date est déterminée par le système et l'entrée d'autres données peut la faire disparaître.
Si l'on suit, au contraire, la méthode présentée en page 602 (voir le listing correspondant pages 604 et 605), l'inscription

../../..

apparaîtra à l'écran, avec le curseur placé sous le premier point.
Entrer les données devient alors aussi simple que remplir un formulaire : les points et les séparateurs (symboles /) indiquent à l'utilisateur la longueur de chaque élément. De plus, l'inscription apparaît à l'endroit voulu de l'écran, et y demeure tant qu'une instruction spéciale du programme ne demande pas son

SOUS-PROGRAMME ANNEE BISSEXTILE

Entrée : ANNEE$ = chaîne contenant l'année.
Sortie : FLAGB = 0 si l'année est bissextile, sinon FLAGB = 1.
Calculs : l'année est bissextile si elle est divisible par quatre.



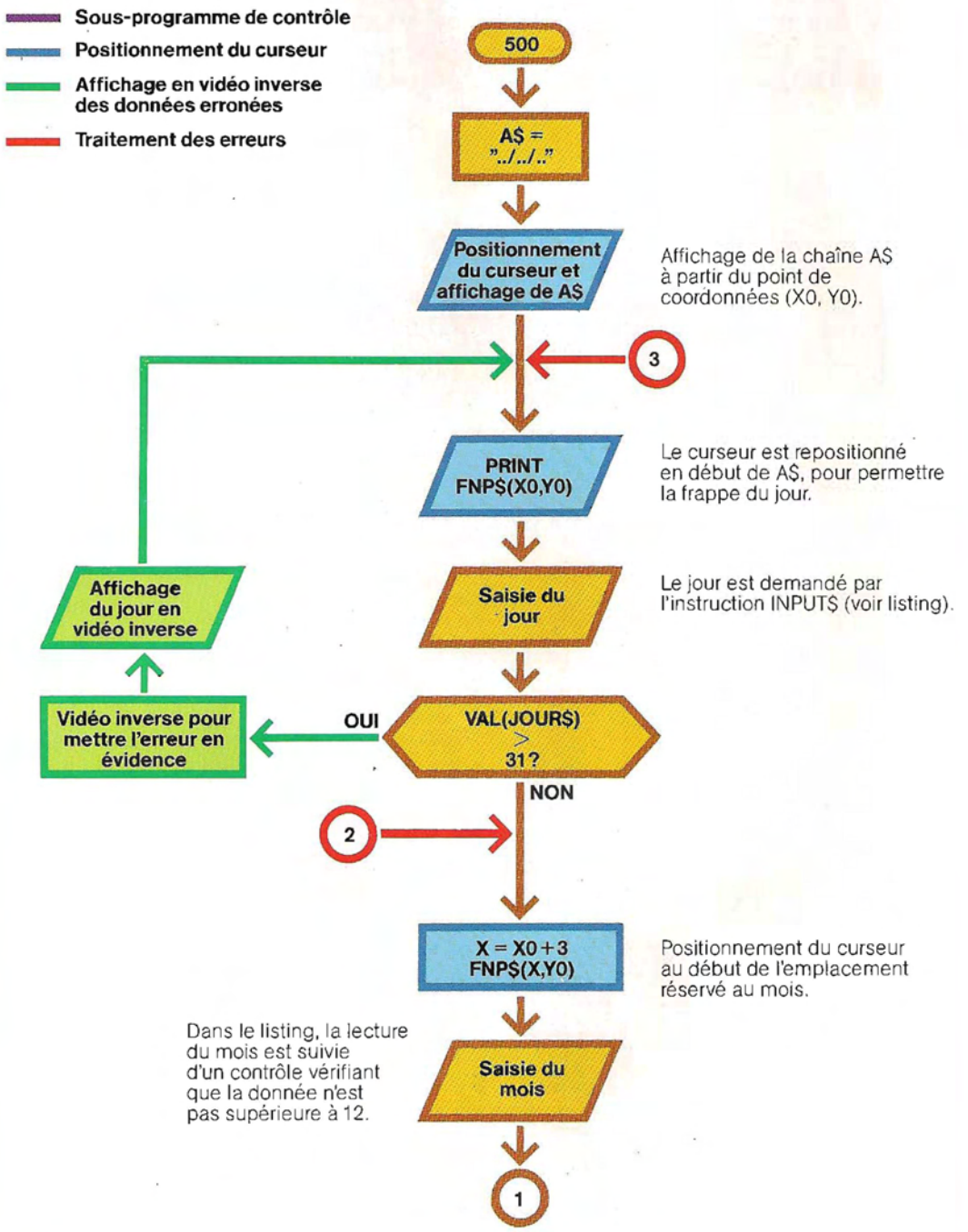
Les années bissextiles sont divisibles par quatre. Toutefois, les années séculaires (1900, 2000, etc.) ne sont bissextiles que si elles sont divisibles par quatre cents.

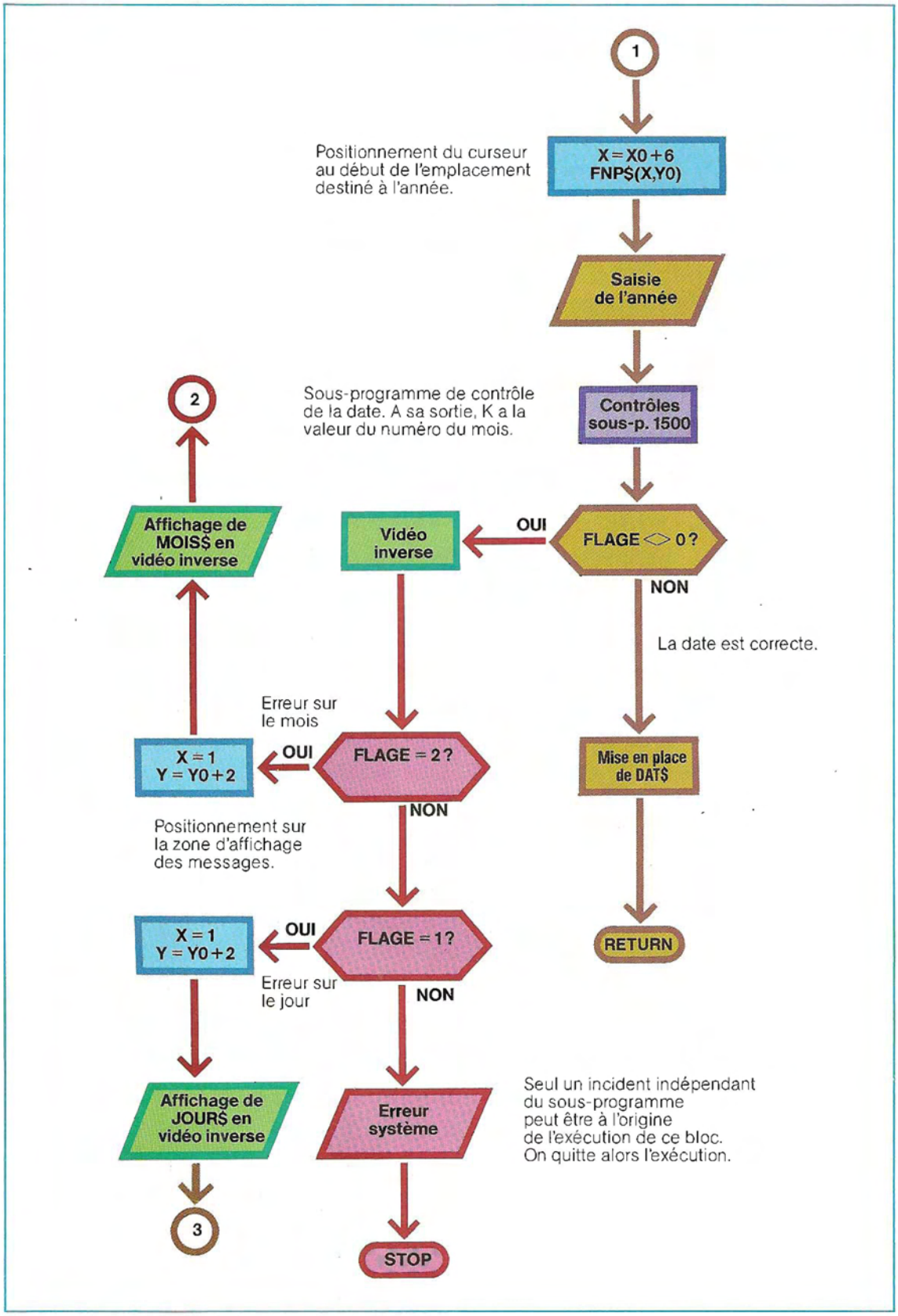
SOUS-PROGRAMME DE SAISIE ET DE CONTROLE DE LA DATE

On utilise, dans ce sous-programme, deux tableaux remplis par des instructions DATA :
M$(12) contient les numéros des mois (1, 2, 3, etc.) ;
NJ(12) mémorise le nombre de jours correspondants (31, 28, 31, etc.) classés dans le même ordre.

Entrées : (X0, Y0) = coordonnées du point de l'écran à partir duquel s'affiche la date.

Sortie : DAT$ = chaîne de caractères contenant le jour, le mois et l'année (tous exprimés par deux chiffres).




1
Positionnement du curseur au début de l'emplacement destiné à l'année.
X=X0+6
FNP$(X,Y0)
Saisie de l'année
Contrôles sous-p. 1500
Sous-programme de contrôle de la date. A sa sortie, K a la valeur du numéro du mois.
FLAGE <> 0 ?
OUI
Vidéo inverse
Affichage de MOIS$ en vidéo inverse
2
NON
La date est correcte.
Mise en place de DAT$
RETURN
Erreur sur le mois
FLAGE = 2 ?
OUI
X=1
Y=Y0+2
Positionnement sur la zone d'affichage des messages.
NON
FLAGE = 1 ?
OUI
X=1
Y=Y0+2
Erreur sur le jour
NON
Affichage de JOUR$ en vidéo inverse
3
Erreur système
STOP
Seul un incident indépendant du sous-programme peut être à l'origine de l'exécution de ce bloc. On quitte alors l'exécution.

PROGRAMME DE SAISIE ET DE CONTROLE DE LA DATE

```
10 ' **** PROGRAMME TEST ****
15 ' FICHIER = DATE
30 DEFINT A-Z
30 DEF FNP$(X,Y)=CHR$(27)+CHR$(61)+CHR$(31+Y)+CHR$(31+X)
40 DEF FNN$=CHR$(27)+"G0"
50 DEF FNI$=CHR$(27)+"G4"
60 DIM M$(12),NJ(12)
62 FOR I=1 TO 12 : READ M$(I) : NEXT I
64 DATA"01"
66 DATA"02"
68 DATA"03"
70 DATA"04"
72 DATA"05"
74 DATA"06"
76 DATA"07"
78 DATA"08"
80 DATA"09"
82 DATA"10"
84 DATA"11"
86 DATA"12"
87 X0=5 : Y0=10
88 FOR I=1 TO 12 : READ NJ(I) : NEXT I
90 DATA 31,28,31,30,31,30,31,31,30,31,30,31
91 'PRINT CHR$(27)+"+"          ' EFFACE L'ECRAN
92 GOSUB 500          ' SAISIE DE LA DATE ET CONTROLE
94 PRINT "La date ";DAT$;" est correcte"
100 INPUT "VOULEZ-VOUS CONTINUER (OUI/NON) ";REP$
110 IF REP$="NON" GOTO 120
115 Y0=Y0+2 : GOTO 92
120 END
500 ' * Sous-programme de saisie de la date *
501 X=1 : Y=Y0-2 : PRINT FNP$(X,Y);      ' Positionne le curseur en (1,8)
505 PRINT "TAPEZ LA DATE SOUS LA FORME ../../.."
510 A$="../../.."
520 PRINT FNP$(X0,Y0);A$          ' Affiche A$ à partir de (5,10)
530 PRINT FNN$                    ' Affichage normal
540 PRINT FNP$(X0,Y0);            ' Positionnement du curseur sur le jour : (5,10)
550 C$=INPUT$(1) : PRINT C$;      ' PREMIER CARACTERE
560 D$=INPUT$(1) : PRINT D$;      ' SECOND CARACTERE
570 JOUR$=C$+D$
580 IF VAL(JOUR$)<=31 THEN GOTO 640     ' On peut continuer si le jour n'est pas
590 ' **** ERREUR : JOUR > 31 ****                    supérieur à 31
600 PRINT FNI$                ' Passage en vidéo inverse
610 PRINT FNP$(X0,Y0);        ' Positionnement au début du jour
620 PRINT JOUR$               ' Le jour est réaffiché en vidéo inverse
630 GOTO 530
640 ' LE JOUR EST A PRIORI CORRECT
650 X=X0+3
655 PRINT FNP$(X,Y0);         ' Positionnement sur le mois : (8,10)
660 C$=INPUT$(1) : PRINT C$;
670 D$=INPUT$(1) : PRINT D$;
680 MOIS$=C$+D$ : IF VAL(MOIS$)<=12 THEN GOTO 690
685 PRINT FNI$ ; PRINT FNP$(X,Y0); MOIS$ : PRINT FNN$ : GOTO 655    ' Correction
690 X=X0+6 : PRINT FNP$(X,Y0);    ' Positionnement sur l'année : (11,10)
700 C$=INPUT$(1) : PRINT C$;
710 D$=INPUT$(1) : PRINT D$;
720 ANNEE$=C$+D$
730 ' ** CONTROLES **
740 GOSUB 1500
750 IF FLAGE=0 GOTO 960       ' Date correcte
760 '
770 ' **** ERREURS ****
780 PRINT FNI$
790 IF FLAGE<>2 GOTO 840
795 '
```

```
800 ' ** ERREUR SUR LA VALEUR DU MOIS **
810 X=1 : Y=Y0+2
820 PRINT FNP$(X,Y); : PRINT MOIS$ : PRINT FNN$ : GOTO 650     ' Relecture du mois
830 '
840 IF FLAGE=1 GOTO 880
850 PRINT "****** ERREUR SYSTEME ******"
860 STOP
870 '
880 ' ** ERREUR SUR LA VALEUR DU JOUR **
890 X=1 : Y=Y0+2
900 PRINT FNI$ : PRINT FNP$(X,Y); : PRINT JOUR$
910 GOTO 530        ' Relecture du jour et du mois
920 '
930 ' **** DATE CORRECTE ****
940 DAT$=JOUR$ + " " + MOIS$ + " " + ANNEE$
950 RETURN
960 '
1500 ' * CONTROLE DE COMPATIBILITE ENTRE LE JOUR ET LE MOIS *
1510 K=0 : FLAGE=0 : FLAGB=0
1520 FOR I=1 TO 12
1530 IF MOIS$=M$(I) THEN K=1
1540 NEXT I
1550 '
1560 J=VAL(JOUR$)
1570 IF K<>2 OR J<>29 GOTO 1630
1580 ' * Dans ce cas, J=29 et K=2 *
1590 GOSUB 2000     ' Sous-programme année bissextile
1600 IF FLAGB=1 GOTO 1650
1610 GOTO 1660
1620 ' * Dans ce cas, K<>2 ou J<>29 *
1630 IF K=0 THEN FLAGE=2 : GOTO 1660     ' Erreur de mois
1640 IF J<=NJ(K) GOTO 1660
1650 FLAGE=1        ' Erreur de jour
1660 RETURN
1670 '
2000 ' * SOUS-PROGRAMME ANNEE BISSEXTILE *
2010 V=VAL(ANNEE$)
2020 V1=V MOD 4
2030 '
2040 IF V1=0 GOTO 2060     ' Année bissextile
2050 FLAGB=1               ' Année non bissextile
2060 RETURN
```

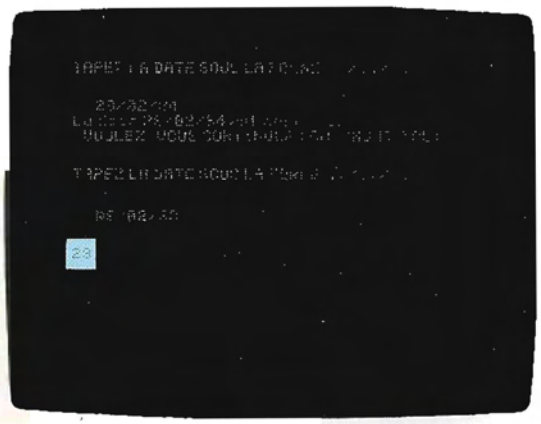
effacement. On peut très bien prévoir l'entrée d'autres données dans une autre partie de l'écran.
On a complété le sous-programme par des instructions qui attirent l'attention de l'opérateur sur les données incorrectes, en les affichant en vidéo inverse.
Cette méthode est très souvent employée dans les programmes qui demandent la saisie d'un grand nombre de données, car elle permet à l'opérateur de voir son erreur du premier coup d'œil.

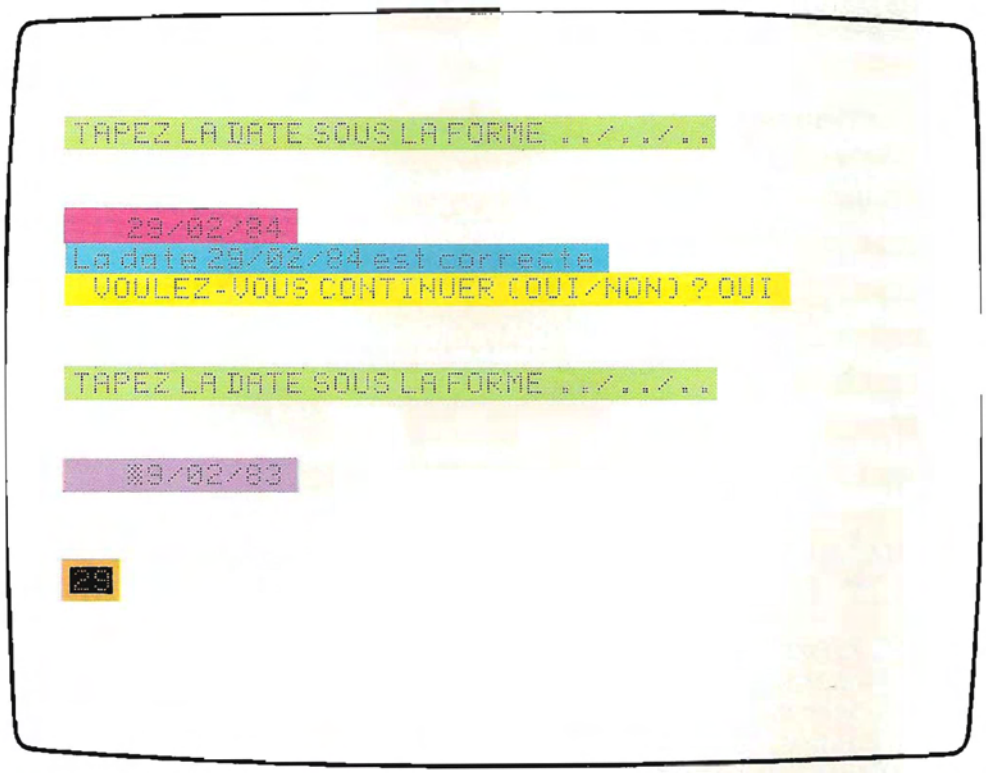
Les touches de fonction

De nombreux ordinateurs possèdent des touches programmables. On les appelle « touches de fonction » car leur frappe déclenche l'exécution de fonctions programmées.
La frappe d'une touche génère le code ASCII correspondant. La touche marquée A est ainsi associée à 65 (nombre décimal), celle marquée B à 66 et ainsi de suite. Aucun code n'est, au contraire, attribué a priori aux touches de fonction. Le programmeur dispose d'instructions pour leur associer lui-même telle ou telle valeur, en fonction de ses besoins.
Les touches de fonction sont réservées aux commandes. On leur affecte donc des codes qui ne sont pas utilisés en ASCII.
On peut ainsi attribuer à une touche de fonction le nombre 18, qui, en ASCII, ne correspond ni à une lettre, ni à un chiffre. Le système reconnaîtra alors ce code et pourra interrompre une opération en cours pour exécuter la commande associée. On pourrait, par exemple, faire correspondre la sauvegarde sur disque au code 18, l'impression au code 19, etc. L'organigramme ci-dessous illustre la méthode à employer : il utilise trois touches de fonction, qui ont été associées aux valeurs 18, 19 et 20.
Il faut, bien entendu, attribuer explicitement à

SAISIE ET CONTROLE DE LA DATE

Cette page retrace l'exécution du programme DATE (pages 604 et 605).





L'instruction 505 commande l'affichage de cette phrase, destinée à guider l'utilisateur.

L'exécution de la ligne 520 fait apparaître à l'écran le masque de saisie ../../.. La photographie a été prise alors que l'opérateur avait déjà tapé la date.

L'entrée de la date est suivie du contrôle de validité prévu dans le programme. 1984 étant bissextile, la subroutine vérifie que la date 29/02/84 est correcte, et en avertit l'opérateur (ligne 94).

Le programme (ligne 100) demande à l'opérateur s'il désire tester une nouvelle date. Celui-ci répond par l'affirmative, et l'exécution reprend en ligne 92.

Cette fois, la date entrée est erronée. Il ne peut, en effet, y avoir de 29 février 1983 puisque l'année 1983 n'est pas bissextile.

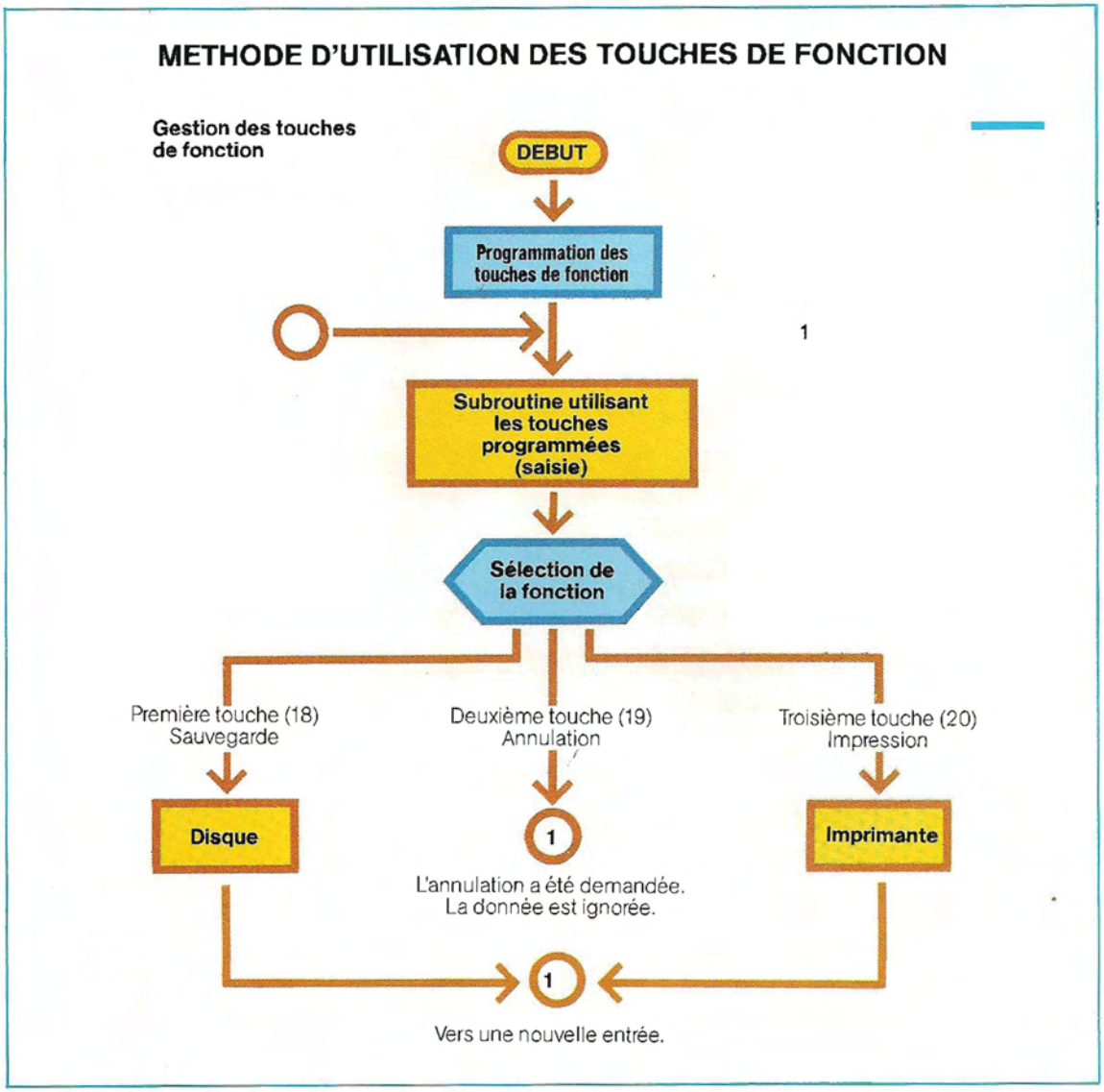
Cette erreur est détectée par le contrôle de compatibilité jour/mois. Le numéro du jour apparaît donc en vidéo inverse (instructions 890 et 900) tandis que le curseur se repositionne sur le premier chiffre de ce numéro (lignes 530 et 540).

chaque touche la valeur numérique voulue, avant de pouvoir l'utiliser. Les programmes se servant de touches de fonction doivent donc inclure un module d'initialisation des codes.
La programmation d'une touche de fonction conserve son effet jusqu'à l'extinction de la machine, à moins qu'on ne la modifie par une nouvelle affectation.
L'identification des touches de fonction n'obéit pas aux mêmes critères sur toutes les machines. Dans l'exemple que nous allons développer, les touches sont désignées par des lettres minuscules.
Sur un certain nombre de machines, le module d'initialisation est constitué de la séquence suivante :

- Escape = CHR$(27)
- Fonction d'initialisation = CHR$(46)
- Identification de la touche (pour la touche c, par exemple, CHR$(99) puisque c = 99 en ASCII).
- Valeur hexadécimale à associer à la touche, sous la forme de deux caractères en représentation ASCII.

Ainsi, le code ASCII de la lettre A étant 41 (en hexadécimal), il faut, pour associer la lettre A à une touche de fonction, transmettre séparément les caractères ASCII 4 et 1.
En décimal, le code de 4 est 52 et celui de 1 est 49. On représentera donc la lettre A par les caractères CHR$(52) et CHR$(49).

METHODE D'UTILISATION DES TOUCHES DE FONCTION



Voici la chaîne complète nécessaire à l'initialisation de la touche c :

A$=CHR$(27) + CHR$(46) + CHR$(99) + CHR$(52) + CHR$(49)

- CHR$(27) : Escape
- CHR$(46) : Code d'initialisation
- CHR$(99) : Identification de la touche (c)
- CHR$(52) + CHR$(49) : Caractères représentant la valeur (A) à associer à la touche.

Ensuite, une simple instruction PRINT A$ permettra de rendre la touche opérationnelle. Remarquons que dans ce cas, le mot « PRINT » ne provoque aucune impression (de même que PRINT CHR$(7) émet un « bip »).
Le listing d'un programme qui affecte les valeurs 17, 18 et 19 à trois touches de fonctions (c, d et e) est reproduit ci-dessous, à titre d'exemple.
L'acquisition des données avec masques de saisie écran constitue l'une des applications des touches de fonctions programmables. Les pages 609 et 610 reproduisent l'organigramme général d'un module de saisie type. Le détail de ce module sera développé un peu plus loin. Pour l'instant, notons simplement qu'il effectue les opérations suivantes :

1 / affichage, en un point quelconque de l'écran, du libellé de la donnée à saisir. Visualisation de la longueur de cette donnée par des pointillés ;

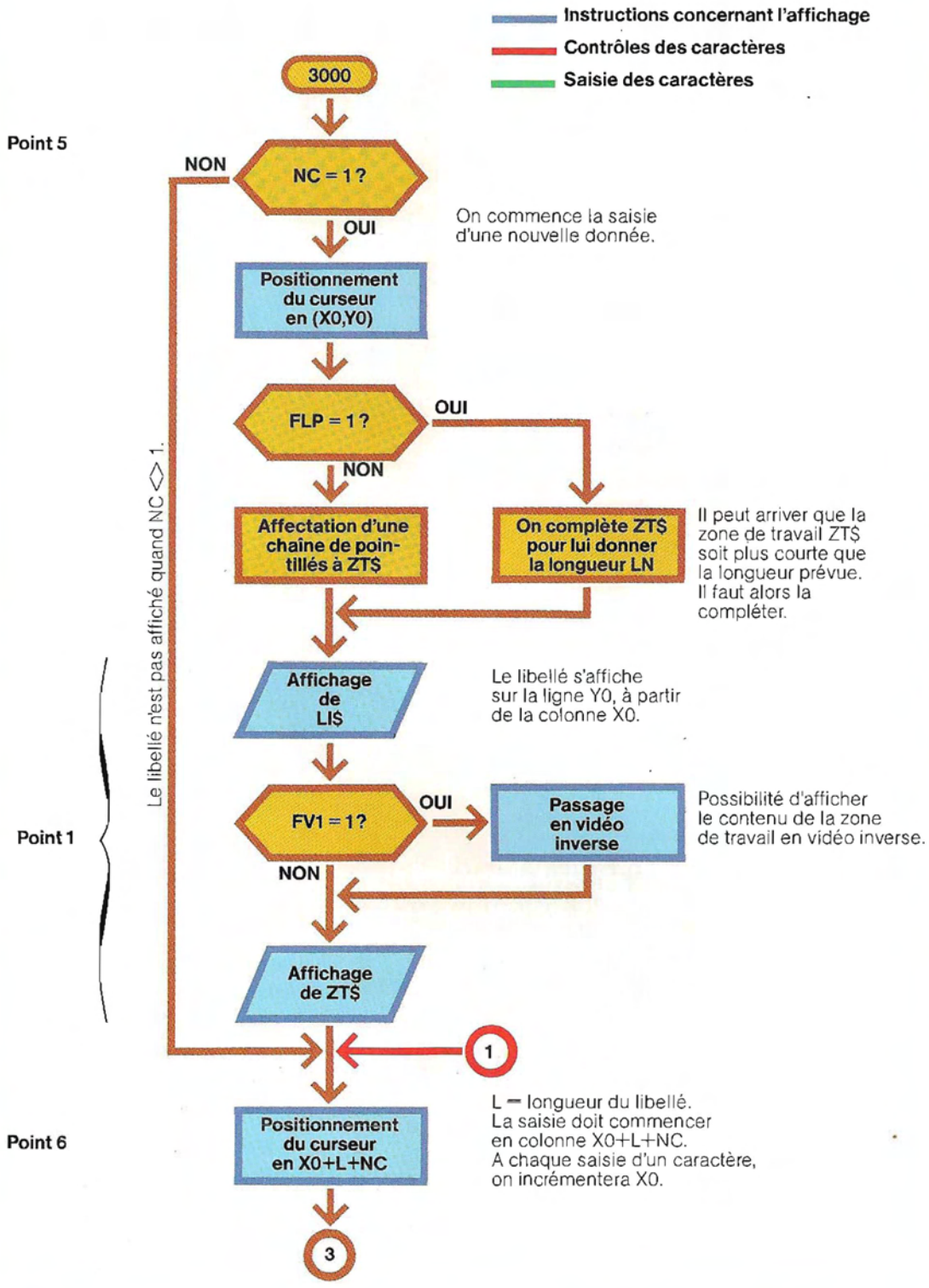
2 / contrôle du type de chaque caractère entré. Si TP vaut 1, la donnée attendue doit être

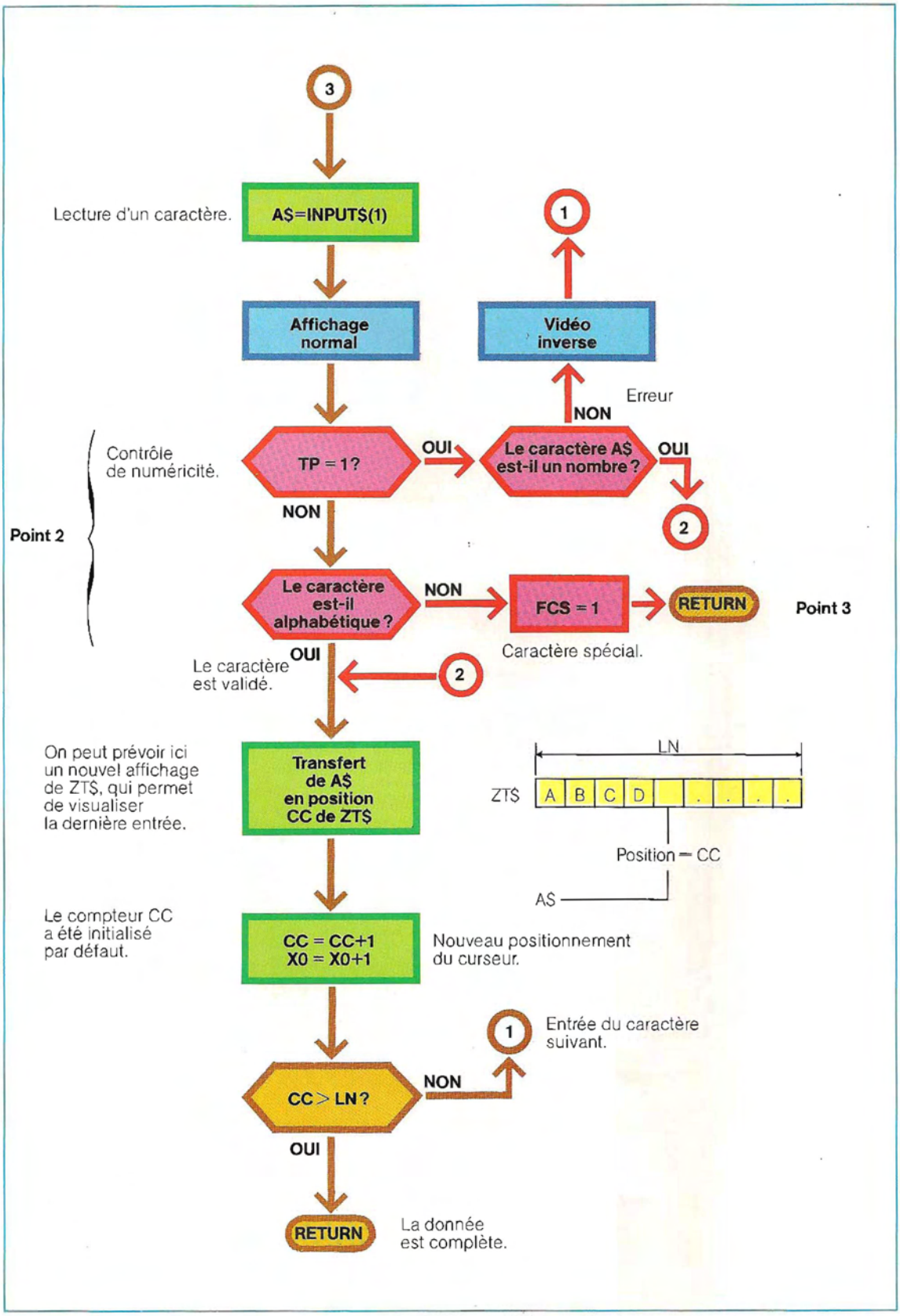
EXEMPLE DE PROGRAMMATION DES TOUCHES DE FONCTION

```
10 ' *** EXEMPLE DE PROGRAMMATION DES TOUCHES DE FONCTION ***
15 '     FICHIER TFN
30 DEFINT A-Z
40 A$=CHR$(27)+CHR$(46)       ' PARTIE COMMUNE DE LA CHAINE
50 B3$=CHR$(99)               ' TOUCHE N.2 = c
60 B4$=CHR$(100)              ' TOUCHE N.3 = d
70 B5$=CHR$(101)              ' TOUCHE N.4 = e
71 ' ** ATTRIBUTION DE LEUR VALEUR AUX TOUCHES DE FONCTION **
72 '    Les valeurs doivent etre exprimees en hexadecimal
73 '    Exemple : 11 hexadecimal equivaut a 17 decimal
74 '    Si l'on attribue cette valeur a une touche,
75 '    on obtiendra 17 par la fonction ASC
80 C1$=CHR$(49)+CHR$(49)       ' TOUCHE N.2 = 17
90 C2$=CHR$(49)+CHR$(50)       ' TOUCHE N.3 = 18
92 C3$=CHR$(51)+CHR$(51)       ' TOUCHE N.4 = 19
110 PRINT A$+B3$+C1$               ' PROGRAMMATION
120 PRINT A$+B4$+C2$               '       DES
130 PRINT A$+B5$+C3$               '    TOUCHES
140 '
150 PRINT "Enfoncez une touche quelconque"
160 I$=INPUT$(1)
162 ' Pour connaitre la valeur correspondant a la touche frappee,
163 ' on utilise la fonction ASC(I$). On obtient l'equivalent
164 ' decimal de la valeur hexadecimale attribuee a
165 ' la touche
170 IF ASC(I$)<17 OR ASC(I$)>19 THEN GOTO 260    ' On n'a programme que
172 '                                              les valeurs 17, 18 et 19
180 K=ASC(I$)-16       ' K vaut 1, 2 ou 3 selon la touche enfoncee
190 ON K GOTO 200, 220, 240
200 NU=2 : GOTO 250
220 NU=3 : GOTO 250
240 NU=4
250 PRINT "VOUS AVEZ APPUYE SUR LA TOUCHE DE FONCTION N.";NU
255 GOTO 270
260 PRINT "VOUS N'AVEZ PAS APPUYE SUR UNE TOUCHE DE FONCTION PROGRAMMEE"
270 INPUT "VOULEZ-VOUS CONTINUER ";REP$
280 IF REP$="OUI" THEN GOTO 150
300 END
```

SUBROUTINE D'ACQUISITION DES DONNEES AVEC MASQUE DE SAISIE

L'alignement à droite ou à gauche (point 4) ne fait pas partie de ce module : il vaut mieux le traiter séparément.




3
Lecture d'un caractère.
A$=INPUT$(1)
Affichage normal
1
Vidéo inverse
Erreur
NON
Contrôle de numéricité.
TP = 1?
OUI
Le caractère A$ est-il un nombre ?
OUI
2
Point 2
NON
Le caractère est-il alphabétique ?
NON
FCS = 1
RETURN
Point 3
Caractère spécial.
OUI
Le caractère est validé.
2
On peut prévoir ici un nouvel affichage de ZT$, qui permet de visualiser la dernière entrée.
Transfert de A$ en position CC de ZT$
LN
ZT$
A B C D
Position = CC
A$
Le compteur CC a été initialisé par défaut.
CC = CC+1
X0 = X0+1
Nouveau positionnement du curseur.
1
Entrée du caractère suivant.
CC > LN ?
NON
OUI
RETURN
La donnée est complète.

numérique ; en cas d'erreur, le curseur n'avance pas ; le caractère erroné s'affiche en vidéo inverse et on redemande une saisie. Si TP est différent de 1, le caractère doit être alphabétique ;

3 / lorsque le caractère entré n'est ni une lettre, ni un chiffre, son code numérique est mémorisé dans une variable, on positionne un flag (FCS) à 1 et on sort du sous-programme. Cette méthode permet de faire intervenir des touches de fonction programmables, gérées dans une autre routine ;

4 / quand une donnée ne remplit pas la zone qui lui est allouée, elle est alignée à droite ou à gauche, selon qu'elle est numérique ou alphanumérique, puis réaffichée. Ce point, traité séparément, n'apparaît pas dans notre organigramme ;

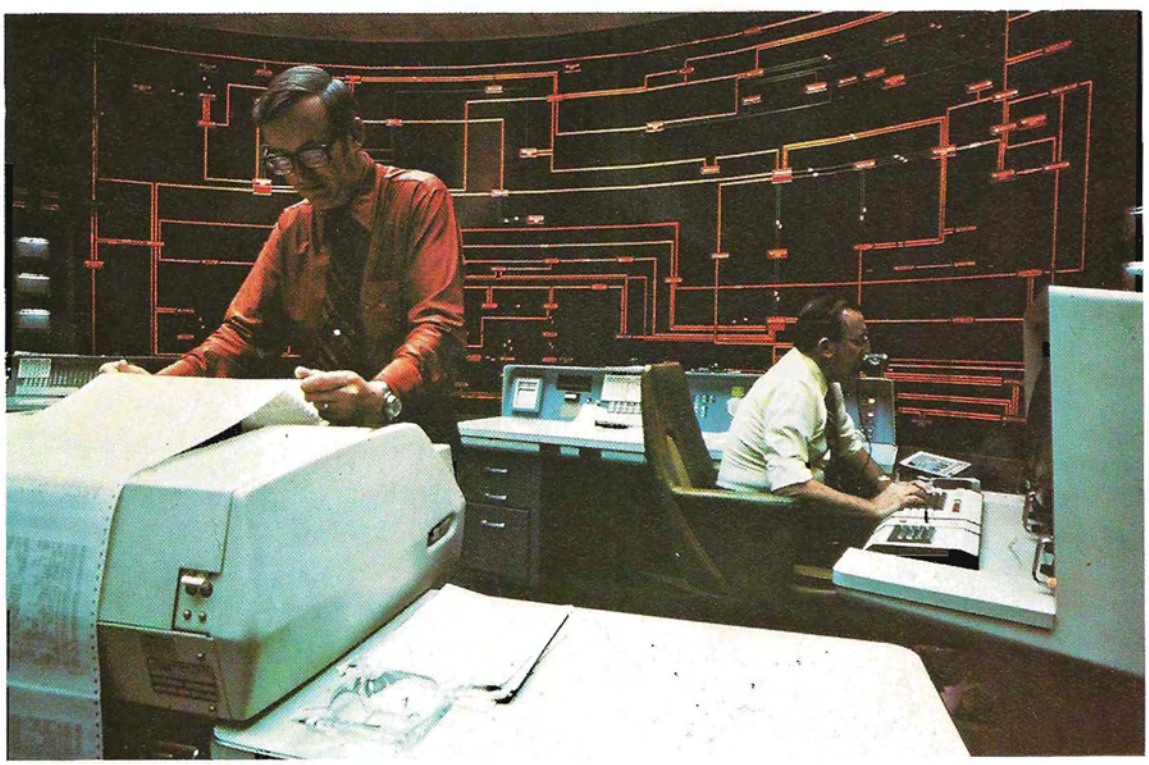
5 / les données transitent par une zone de travail, dont l'affichage est subordonné à la valeur d'un flag (NC). Lorsqu'on appelle ce sous-programme, on peut donc afficher une donnée à compléter (NC + 1) ou alors (NC = 1), afficher un libellé (indiquant qu'on attend la saisie d'une nouvelle donnée), suivi de pointillés ;

6 / la position initiale du curseur (où commencera la saisie) doit être déterminée dans le programme d'appel si on veut utiliser cette routine pour une saisie partielle.

Le rôle des points 5 et 6 sera éclairci par un exposé plus approfondi des masques de saisie.
Voici la liste des variables utilisées dans ce sous-programme :

ZT$ = Zone de travail où les données sont stockées temporairement ;
LN = longueur de la zone allouée à la donnée (en caractères) ;
TP = type de la donnée attendue (1 = numérique, 2 = alphanumérique) ;
LI$ = chaîne contenant le libellé de la donnée ;
X0,Y0 = position du curseur (coordonnées) sur l'écran ;
NC = numéro du caractère à partir duquel doit commencer la saisie ;

La salle de contrôle d'une centrale électrique moderne.



K. Reese/Marka

Informatique et vie privée

Banques de données : *voilà l'un des termes les plus répandus du jargon informatique. Il faut dire qu'elles représentent aujourd'hui un organe essentiel de la direction des institutions gouvernementales et des établissements publics ou privés. Dans les pays industrialisés, aucun citoyen ne peut désormais se vanter de n'être recensé dans aucun fichier informatisé. Toutefois, cette « mise en fiches » ne mérite pas d'être systématiquement dénigrée. Sans elle, les administrations seraient impuissantes, les services d'action sanitaire et sociale ne pourraient remplir leur rôle ; comment planifier les impôts, déterminer l'utilité des travaux publics et mener une politique d'urbanisme cohérente sans des recensements nationaux ?*

Il s'avère cependant légitime de s'inquiéter de certains abus. C'est ainsi qu'aux Etats-Unis, on ne compte pas moins de deux mille établissements de crédit qui recueillent d'énormes quantités d'informations confidentielles. Pour ne citer qu'un exemple, 100 millions de dossiers sont rassemblés dans les fichiers de l'un d'eux.

Pourtant, bien plus que la quantité des informations recueillies, la nouveauté réside dans la façon dont elles sont mémorisées.

Il y a bien longtemps, en effet, que l'on rassemble des informations, mais on les entassait autrefois dans des archives réparties sur d'interminables rangées d'étagères, où il n'était pas toujours facile de les retrouver. La disponibilité des informations s'est considérablement accrue avec l'apparition des mémoires de masse à support magnétique, de capacité bien supérieure et d'accès beaucoup plus aisé.

De plus, la notion de distance n'a plus guère de sens puisqu'on peut les interroger depuis des terminaux très éloignés. C'est ainsi qu'il n'y a pas si longtemps, les dossiers de sécurité sociale du Canada étaient gérés dans un centre situé en Corée.

Il faut reconnaître qu'il est pratiquement impossible pour le particulier de connaître l'existence et la finalité des données enregistrées sur son compte et de savoir qui y a accès. C'est pourquoi on cherche, en dépit de nombreux obstacles, à mettre en place des conventions internationales. De nombreux défenseurs des libertés ont cherché à protéger la vie privée du citoyen, en faisant reconnaître le droit de chacun au contrôle de la divulgation des informations qui le concernent.

Tout citoyen est désormais contraint de fournir des renseignements personnels à différents ***services administratifs*** *qui les engrangent dans leurs banques de données. Il est impossible d'ouvrir un compte ou de conduire une automobile sans se plier à cette obligation. Il faudrait cependant limiter le volume d'informations conservées dans ces banques et leur dissémination. Il semble, en outre, qu'à l'époque de l'ordinateur, le sérieux et l'exactitude des informations nominatives ne soient pas garantis comme on pourrait le souhaiter. Il arrive, en effet, que des renseignements soient inutiles, incomplets, périmés, voire complètement erronés. Plus grave encore, des informations fournies dans un but précis pourraient être communiquées à d'autres services à des fins totalement différentes. Enfin, sans l'institution d'un « droit de regard », comment s'assurer que des renseignements ne sont pas recueillis illégalement et à l'insu des citoyens !*

Les fichiers des ***services de police*** *contiennent, d'ores et déjà, un nombre d'informations impressionnant. Même dans les pays les plus démocratiques, les ordinateurs de la police ingurgitent et classent les informations les plus diverses, que ce soit des listes de véhicules volés, d'automobilistes indisciplinés, de propriétaires de véhicule, ou des renseignements sur des personnes faisant l'objet d'un avis de recherche, en passant par le signalement anthropométrique et les empreintes digitales des criminels. On constitue aussi parfois des fichiers concernant les personnes simplement suspectées d'activités politiques clandestines ou terroristes. En Grande-Bretagne, toute personne qui acquiert un véhicule doit le faire immatriculer au centre de Swansea. Transmis à l'ordinateur central de la police, les renseignements ainsi réunis vont alimenter un fichier où sont déjà recensés 26 millions de citoyens, dont la plupart n'ont vraiment rien à se reprocher.*

Comme nous l'avons vu, il arrive que des informations fournies à une administration

pour satisfaire une obligation légale soient divulguées pour servir à tout autre chose. Enfin, il est de notoriété publique que le secret des informations détenues par la police est particulièrement mal gardé, puisque les détectives des agences privées – souvent dirigées par d'anciens membres de la police – conservent leurs facilités d'accéder aux fichiers qui les intéressent.
Des criminels, qui connaissent un numéro de téléphone confidentiel, ont même réussi à obtenir des renseignements, en se faisant passer pour des policiers.

Il est très difficile de préserver des indiscrétions les informations conservées dans les banques de données.
Pour chercher une information dans les archives traditionnelles, il fallait qu'une personne ouvre une pièce ou une armoire. Il était donc possible de protéger les fichiers par des serrures de sécurité. De plus, si quelqu'un parvenait, malgré tout, à déjouer cette protection, il ne pouvait recopier, photographier ou emporter qu'un nombre limité de documents. Au contraire, et malgré l'extrême sophistication de l'informatique, on n'a pas encore inventé l'ordinateur qui saurait empêcher un escroc, décidé et... possédant une solide expérience d'informaticien, d'accéder à un fichier et de s'emparer de son contenu en une seule fois.
Certains de ces délits ont été portés à la connaissance du public et ont fait beaucoup parler d'eux. Ainsi, des étudiants du Massachusetts Institute of Technology ont réussi à se procurer des informations de la plus haute importance pour la défense nationale de leur pays. A l'université du Michigan, d'autres étudiants ont détruit plusieurs fichiers après en avoir décrypté les mots de passe. Un passionné d'informatique est parvenu à obtenir, dans les fichiers du FBI, des renseignements sur des personnes suspectes ou arrêtées. Ces « professionnels » de haut vol sont d'ailleurs très recherchés. On en a arrêté quelques-uns, mais ils ont presque tous été engagés à prix d'or par leurs victimes, en tant que conseillers en matière de sécurité !
Par ailleurs, la connection des ordinateurs à des terminaux très éloignés accroît les risques en tous genres. Il suffit d'une interférence entre deux lignes téléphoniques pour qu'une information soit divulguée ou même détruite. Le problème se pose donc en ces termes : que peut-on faire pour ***protéger la vie privée du citoyen****, tout en permettant aux gouvernements et aux administrations de continuer à collecter les informations nécessaires, dans l'intérêt de ce même citoyen ?*
En Grande-Bretagne, les suspects arrêtés dans le cadre des mesures anti-terroristes sont systématiquement photographiés et leurs empreintes digitales sont relevées. Ces informations demeurent ensuite dans les fichiers même s'il n'y a pas de suites judiciaires (et c'est le cas le plus fréquent). On peut ficher les personnes qui ont un lien quelconque avec des criminels connus, ainsi que les personnes qui sont placées sous surveillance parce qu'on les suspecte de s'apprêter à commettre un délit. Tout cela s'insère dans une politique de prévention légitime, dont le but consiste à empêcher les délits ou d'accélérer l'arrestation des coupables.

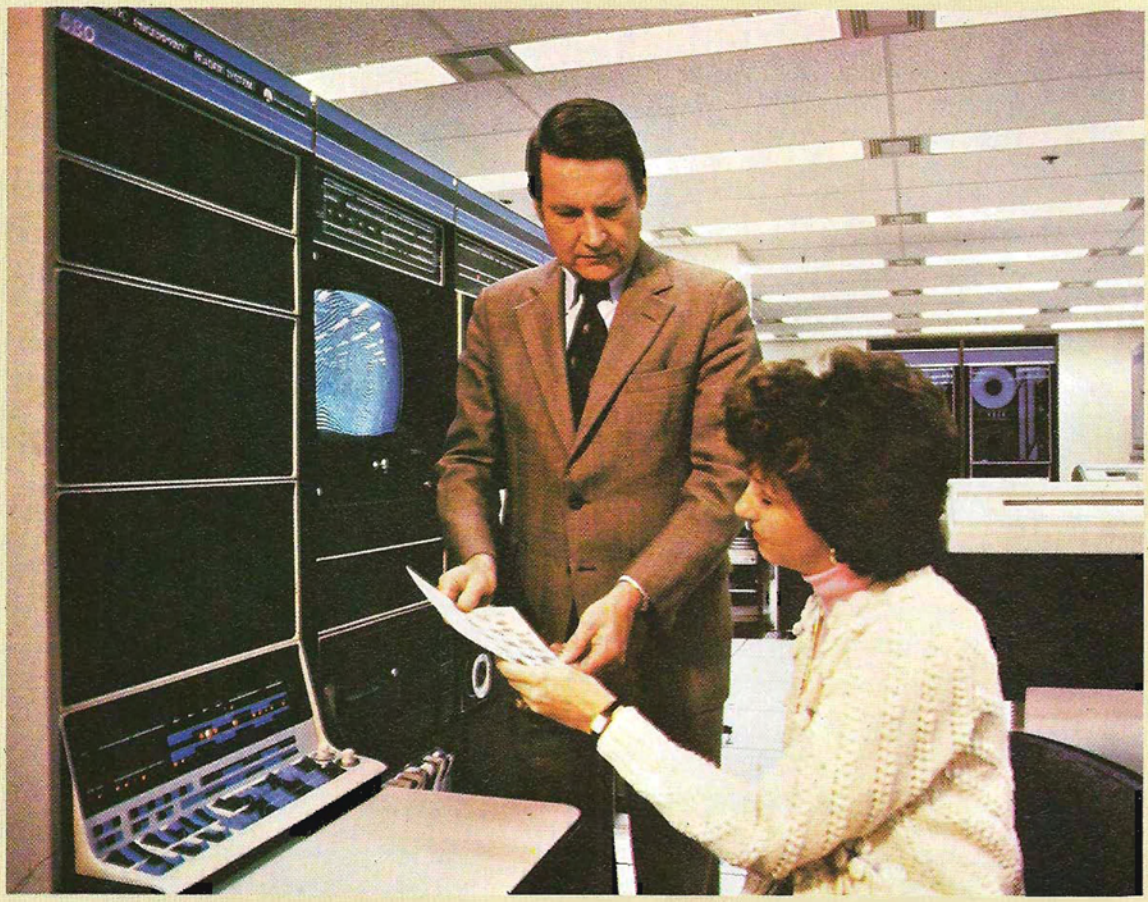
La technologie permet aujourd'hui de corréler, de confronter et de transmettre d'énormes quantités de renseignements que le simple particulier jugerait sans doute tout à fait personnels. Des agences spécialisées mettent des « registres de crédit » à la disposition des responsables d'entreprise qui désirent se renseigner sur la situation financière de leurs clients potentiels. La plus grande banque de données du monde appartient à l'une de ces agences, la TRW Credit Data. Elle rassemble des informations nominatives sur plus de 50 millions de citoyens américains.
Les principales banques d'escompte suédoises ont récemment créé un institut, l'Agence d'Informations pour le Crédit, qui fournit le nom, l'adresse, les revenus, l'état-civil, les activités imposables et la situation judiciaire de tous les citoyens suédois âgés de plus de 16 ans, ainsi que les contrats de location ou de vente qu'ils ont conclus. Cependant, cet institut remet à l'intéressé une copie de tous les documents communiqués à des tiers.
Malheureusement, il est rare que les ***agences de crédit*** *vérifient l'exactitude des informations recueillies. Leur rôle se limite à collecter des renseignements sur les dettes et les créances irrécouvrables et à les communiquer aux organismes qui ont besoin de s'informer avant d'accorder un prêt.*

L'existence de ces fichiers se justifie, parce qu'ils permettent de prévenir l'endettement excessif. Cependant, ils sont parfois à l'origine de graves erreurs. Citons le cas de cet homme d'affaires, à la tête d'une entreprise prospère, qui, désireux de développer son activité, sollicita un prêt et essuya un refus. Bouleversé, il demande une étude sur la situation de son entreprise et il lui fut conseillé de ne pas trop insister. Il approfondit pourtant la question et finit par découvrir qu'une plainte avait été déposée peu auparavant contre l'un de ses homonymes.

Personne ne songe à contester l'importance des ***services sociaux****. Toutefois, on peut regretter que leur organisation fasse peser de lourdes menaces sur les libertés individuelles. En effet, l'inquiétude et le désarroi conduisent de nombreuses personnes à se confier au personnel de ces services.*
Il peut très bien arriver que les informations enregistrées dans une banque de données n'aient pas d'autre fondement que le jugement subjectif d'un fonctionnaire. Dans les rapports des services sociaux, « paranoïaque » et « névrotique » sont deux mots qui qualifient trop souvent les personnes venues demander des secours. Il va de soi, en effet, que la plupart de ces personnes ne viendraient pas demander une aide si elles ne traversaient pas une période d'épuisement nerveux. Le point crucial c'est que ces informations sont mémorisées par l'ordinateur : le simple fait qu'elles soient enregistrées leur donne un caractère de constance et de véracité. Il ne faut pas oublier que les témoignages des services sociaux ont souvent valeur d'expertise auprès des tribunaux et qu'un individu risque d'être étiqueté pour le restant de ses jours, à cause d'un jugement hâtif.
Les informations recueillies au sujet des habitants par les autorités locales sont ventilées dans plusieurs sections, selon le service concerné : services sociaux et financiers, services du logement, de l'éducation et de l'emploi. Les informations à caractère financier abondent dans les fichiers des administrations locales, où sont détenus les registres comptables et où l'on doit connaître la situation financière des particuliers pour accorder, à bon escient, des dizaines d'aides en tout genre.

Ces fichiers sont touchés, en nombre de plus en plus grand, par l'informatisation. Or, s'il demeure théoriquement possible de coder les informations pour en interdire l'accès aux services non compétents, il n'existe ni loi, ni règlement pour empêcher l'échange d'informations entre un bureau et un autre ou même un organisme extérieur comme la police. En fait, avant qu'une personne puisse bénéficier d'une aide, son dossier doit souvent transiter par de multiples bureaux. Mais les gens qui font l'objet de ces enquêtes ne sont pas autorisées à connaître le contenu de leur propre dossier. Vrai ou faux, celui-ci est archivé et parfois étalé au grand jour, au moment le moins propice, alors qu'il est impossible de faire rectifier les erreurs éventuelles ou de demander la justification d'un jugement défavorable.

De nombreux pays sont conscients de la nécessité d'instaurer une législation pour protéger la vie privée, mais cela n'a encore été réalisé que dans quelques-uns. Aux Etats-Unis, de nombreuses mesures ont établi le libre accès aux fichiers nominatifs de l'administration fédérale, ainsi que le droit de contester les informations qu'ils contiennent et de les faire rectifier.
La Suède, où la centralisation et l'informatisation des fichiers sont importantes, a reconnu le droit d'accès de chacun aux dossiers qui le concernent, et une loi réglemente la création et la gestion des banques de données : il faut obtenir une autorisation et se soumettre à des contrôles. En Allemagne fédérale, toute information nominative doit légalement être détruite (ou, au moins, définitivement remisée) au bout de cinq ans. En France et en Hollande, il est interdit de réunir certains types d'informations. C'est ainsi que dans notre pays, il est totalement illégal de faire figurer dans les fichiers des renseignements concernant les origines ethniques, les opinions politiques et les croyances religieuses de qui que ce soit. Au Danemark, la divulgation d'informations concernant la vie privée des individus n'est pas autorisée.
De nombreuses personnes souhaitent l'instauration de contrôles officiels sur le volume, la pertinence et les sources des informations recueillies. Il est certain qu'on ne devrait enregistrer que les données réellement nécessai-



J. Pickerell/Marka

Un exemple parmi d'autres de l'efficacité de l'informatique : les banques de données du FBI, gérées par un PDP 15 de DIGITAL.

res. De plus, les renseignements fournis dans un but précis ne devraient pouvoir être utilisés à d'autres fins sans l'accord de la personne concernée. Il ne devrait s'établir aucun lien entre les différents fichiers nominatifs ; les personnes chargées de la collecte et de la conservation des informations devraient être tenues de respecter un code déontologique, sous le contrôle d'une autorité compétente.

En Grande-Bretagne, la promulgation en 1974 du Consumer Credit Act a mis fin à un certain nombre d'abus. Depuis, il est obligatoire de transmettre aux gens qui déposent une demande de prêt le nom de l'agence auprès de laquelle on se procure des renseignements sur leur situation. Ces personnes peuvent alors se faire communiquer les renseignements fournis, demander, le cas échéant, leur rectification, et même donner leur propre version des faits dans un texte de moins de 200 mots. Cette mesure présente toutefois certaines lacunes. Ainsi, elle n'oblige pas les agences à faire part des corrections aux autres établissements d'où pourraient émaner de nouvelles demandes de renseignements, et elle ne réglemente pas la façon dont elles se procurent leurs informations. Elle n'interdit pas non plus la création et la vente de fichiers injustifiables, contenant, par exemple, des informations sur les idées politiques ou la vie sexuelle des personnes. De surcroît, elle ne limite pas la durée de validité des informations et les dossiers peuvent donc très bien mentionner des dettes remboursées depuis longtemps.

Le même objectif est donc partagé un peu partout dans le monde : protéger les individus contre la divulgation d'informations confidentielles. Toutefois, la rapidité de l'évolution technologique et, notamment, la percée des micro-ordinateurs risquent de donner du fil à retordre au législateur.

CC = compteur du nombre de caractères entrés ;
FCS = ce flag prend la valeur 1 après la saisie d'un caractère spécial (ni alphabétique, ni numérique) ;
FLP = flag d'initialisation. Si FLP vaut 1, le contenu précédent de ZT$ est conservé ; sinon, on lui affecte une chaîne de pointillés ;
FVI = flag d'inversion vidéo ; si FVI est mis à 1 avant l'appel du module, ZT$ sera affiché en vidéo inverse.

On sort du sous-programme dans deux cas :

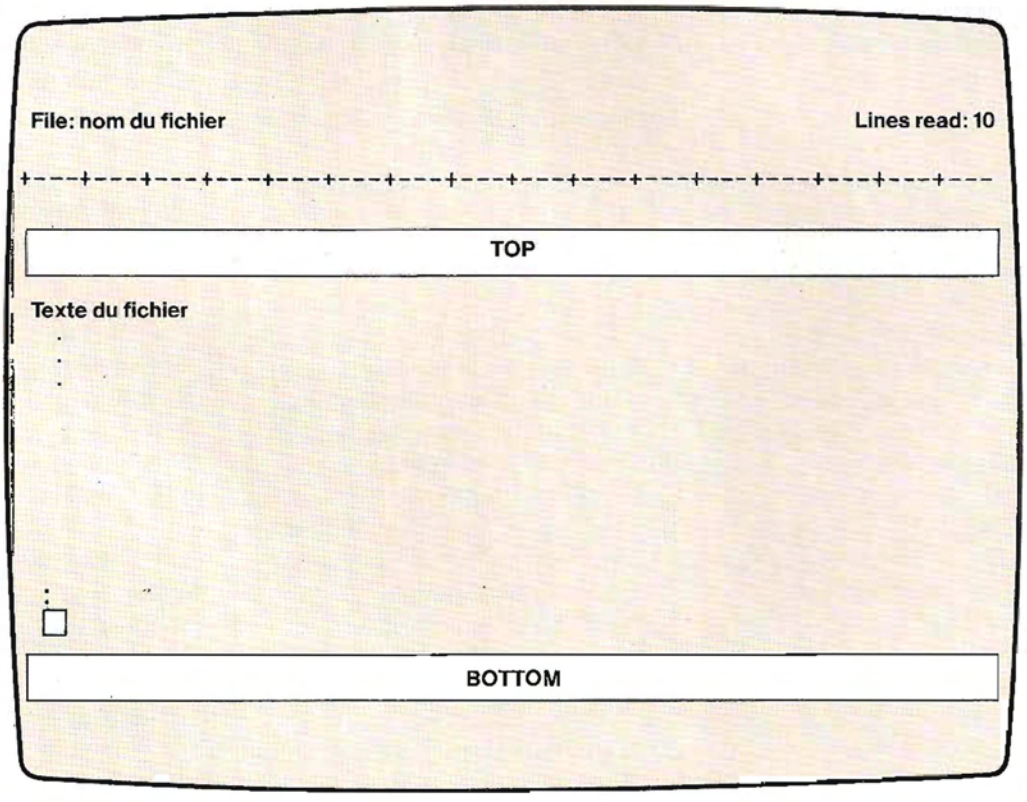
– la donnée est complète, c'est-à-dire que la zone allouée a été remplie (CC = LN) ;
– le caractère saisi n'est ni numérique, ni alphabétique (point 3).
Dans ce dernier cas, le code ASCII du caractère est transmis au programme d'appel ainsi qu'un flag (FCS).

Les éditeurs de texte

On peut tirer parti des multiples possibilités du visuel pour concevoir des programmes spécialement destinés à la préparation des textes ou à la représentation graphique des données. Nous commencerons par l'exemple du Video File Editor*, programme d'application développé par Olivetti pour faciliter l'écriture et la modification des fichiers. Comme la plupart des éditeurs, il s'applique aussi bien aux programmes (sauvegardés en ASCII) qu'aux fichiers de données.
Nous allons décrire les fonctions prévues par ce système, afin de bien montrer l'intérêt qu'il représente.
Tout éditeur permet la création et la modification de fichiers texte, dont les enregistrements sont composés de caractères ASCII imprimables, séparés par les caractères CR,

* Video File Editor est une marque déposée.

STRUCTURE TYPE D'UN FICHIER EN EDITION



LF (Carriage Return = retour chariot et Line Feed = avancement d'un interligne) ou par le caractère RS (Record Separator).
On passe sous éditeur à l'aide de la commande EDIT, suivie d'un nom de fichier. Ce dernier est alors chargé en mémoire, ou créé s'il n'existait pas.
L'éditeur affiche alors dans une « fenêtre » la première partie du fichier. Les dimensions de cette fenêtre varient selon les systèmes : sur le Video File Editor, elle comporte vingt-quatre lignes de quatre-vingts caractères, dont vingt-et-une sont réservées au texte du fichier et les autres à des indications diverses. L'utilisateur peut modifier en mémoire les lignes affichées à l'écran, au moyen des fonctions et des commandes de l'éditeur.
Il peut également remplacer la portion de fichier présente dans la fenêtre par n'importe quelle autre. Quand il a terminé, il sauvegarde sur disque la version modifiée du fichier.
Sous Video File Editor, l'écran se présente comme le montre le schéma de la page 616. Le nom du fichier et sa longueur (en nombre de lignes) s'affichent sur la première ligne.
Rien n'apparaît sur la seconde ligne : elle servira par la suite à l'entrée de commandes ou à la recherche de chaînes.
La largeur de l'écran est découpée en zones de quatre caractères. Les symboles affichés sur la troisième ligne permettent de repérer la position de chaque zone et de chaque colonne. Les vingt-et-une lignes suivantes constituent la fenêtre réservée au texte (text window). La vingt-cinquième ligne de l'écran reste vide. Le début et la fin du texte sont toujours signalés par deux lignes virtuelles, appelées respectivement TOP et BOTTOM (ces mots sont affichés en vidéo inverse). Elles ne font pas partie du texte mais en indiquent simplement les limites.
Quand on passe sous Video File Editor, la ligne TOP apparaît au-dessus de la première ligne du fichier et c'est sur elle que le curseur est positionné. La ligne BOTTOM apparaîtra après la dernière ligne du texte. A sa création, un fichier est vide : on ne voit que les lignes TOP et BOTTOM.
Quant on passe en mode Insertion, le curseur change de forme : il n'apparaît plus comme un tiret placé sous le caractère mais comme un petit triangle situé à sa gauche. Cependant, il clignote dans un cas comme dans l'autre. Le clavier fonctionne différemment quand il est sous le contrôle de l'éditeur de texte. C'est ainsi qu'on peut commander les différentes fonctions d'édition au moyen d'un ensemble de touches programmées automatiquement. Le clavier du M 20 Olivetti est reproduit page 618 et les principales fonctions du Video File Editor sont illustrées pages 619 à 623.
Pour passer en mode commande, l'utilisateur doit enfoncer simultanément la touche COMMAND et la touche 1 (fonction COMMAND MODE). Le curseur se place alors au début de la ligne numéro 2.
A partir de ce moment, toutes les fonctions d'édition de lignes telles que INSERT MODE, BACKSPACE, DELETE CHAR, RECALL LINE s'appliquent à la ligne de commande. La fonction RECALL LINE rétablit, par exemple, son précédent contenu. En revanche, les autres fonctions agissent uniquement sur le fichier texte.
En mode commande, la touche → correspond à la fonction EXECUTE COMMAND et non à la fonction INSERT LINE.
Pour faire revenir le curseur sur le fichier texte et poursuivre la modification de celui-ci, il faut entrer à nouveau la fonction COMMAND MODE.

Les fonctions d'édition et de recherche des chaînes de caractères. Les fonctions de recherche de chaînes permettent de balayer le fichier afin d'y retrouver une suite donnée de caractères.
Avant de commencer la recherche, l'utilisateur doit se mettre en mode commande. Il tape ensuite la chaîne à rechercher sur la seconde ligne de l'écran. Ensuite, il entre la fonction SEARCH DOWN ou la fonction SEARCH UP selon qu'il désire parcourir le fichier vers l'avant ou vers l'arrière par rapport à la position du curseur.
Pour sortir du Video File Editor et revenir sous système, il faut transmettre la commande ABORT (COMMAND + 6) puis CR, ou encore EXIT AND SAVE (CTRL + 6) – qui détermine également la sauvegarde du fichier modifié – puis CR. Si l'on veut sauvegarder le fichier modifié tout en restant sous le contrôle de l'éditeur, il faut entrer SAVE TEXT (CTRL + 5).

Les menus

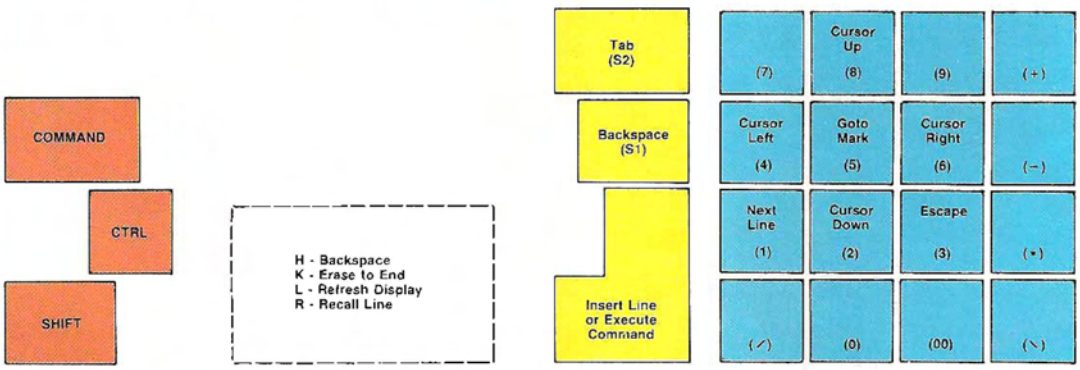
On peut toujours découper une procédure, aussi simple soit-elle, en un certain nombre

LES TOUCHES DE FONCTION DU VIDEO FILE EDITOR

Ces touches numériques sont validées en tant que touches de fonction, ici par la touche COMMAND.

Command Mode (1)	Delete Line (2)	Split Line (3)	Start of Line (4)	Restore Lines (5)	Abort (6)	Insert Mark (7)	Search Up (8)	Line Up (9)	Half Screen Up (0)	Full Screen Up (−)	Top (^)
Insert Mode (1)	Delete Char (2)	Join Lines (3)	End of Line (4)	Save Text (5)	Exit and Save (6)	Reverse Tab (7)	Search Down (8)	Line Down (9)	Half Screen Down (0)	Full Screen Down (−)	Bottom (^)

Et ici, au contraire, par la touche CTRL.



Commandes alphabétiques transmissibles en frappant simultanément la touche CTRL.

Commandes transmissibles en frappant simultanément la touche SHIFT (touche des majuscules).

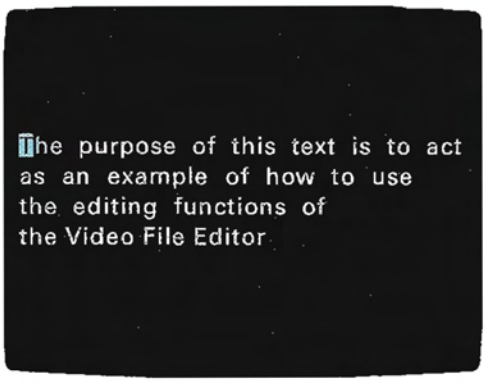
FONCTIONS D'EDITION ET DE RECHERCHE DE CHAINES DE CARACTERES

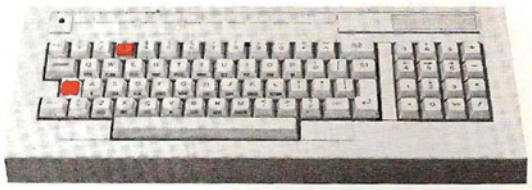
Cette page et les suivantes détaillent l'action des fonctions d'édition et de recherche de chaînes du Video File Editor.
Le fichier texte à modifier est reproduit ci-contre tel qu'il apparaît sur l'écran. Vous trouverez dans la suite deux illustrations pour chaque fonction, celle de gauche indiquant les touches à enfoncer et celle de droite montrant son effet sur le texte.

DELETE LINE
Suppression de la ligne où se trouve le curseur.

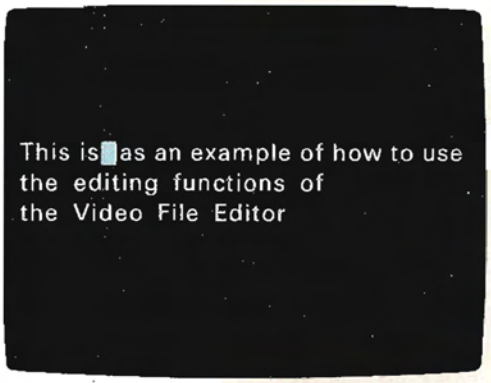
CURSOR UP
Déplacement du curseur d'une ligne vers le haut.

INSERT MODE This is
Insertion de la chaîne indiquée à partir de la position du curseur.

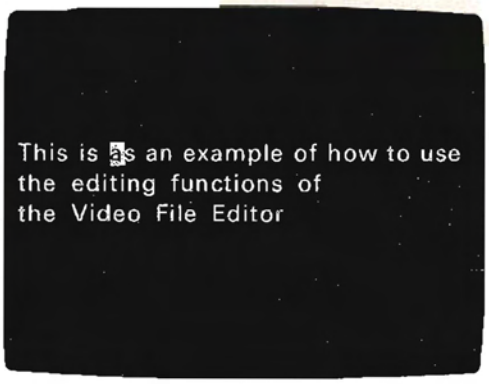




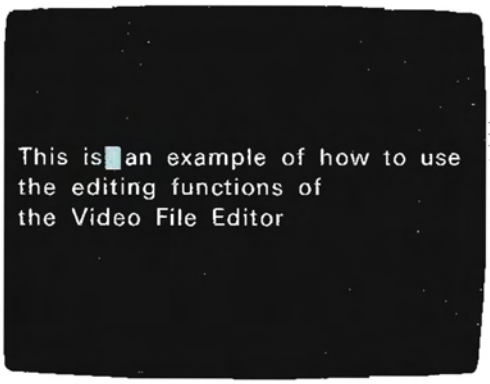
JOIN LINES
Concaténation de la ligne où se trouve le curseur avec la suivante.



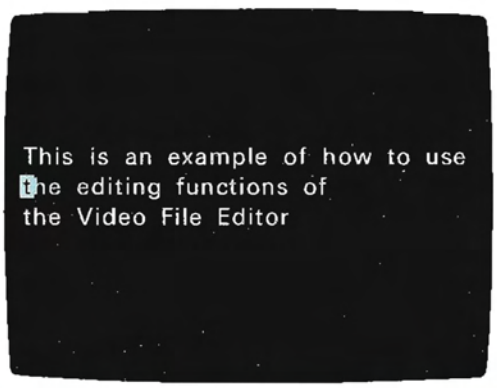
CURSOR RIGHT
Déplacement du curseur d'une colonne vers la droite.



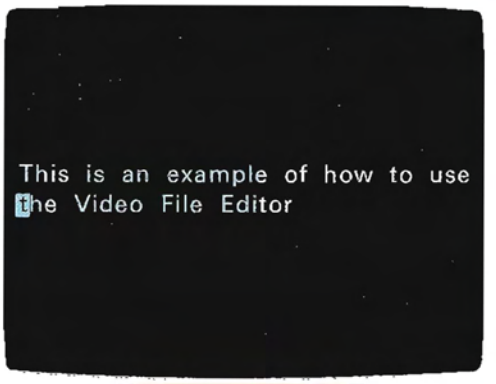
DELETE CHAR
Suppression du caractère sur lequel se trouve le curseur et déplacement d'une colonne vers la gauche de la suite de la ligne.
Ici, la fonction a été utilisée trois fois.



NEXT LINE
Le texte monte d'une ligne et le curseur se place au début de la ligne suivante.

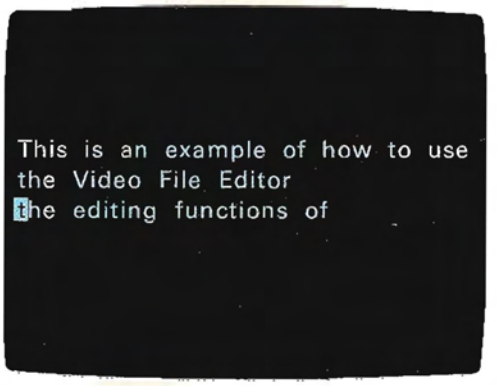


DELETE LINE
Suppression de la ligne où se trouve le curseur.

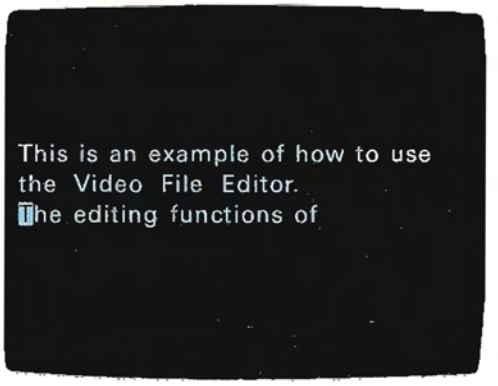


RESTORE LINES
Restitue la ligne précédemment supprimée, sous celle où se trouve le curseur.

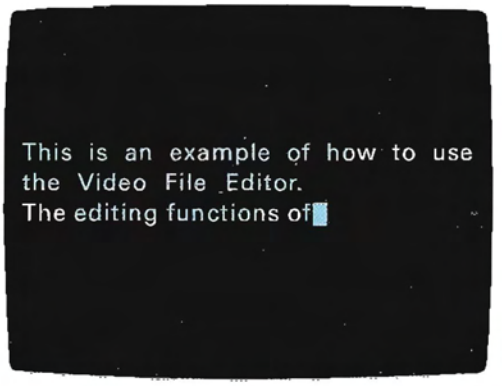
NEXT LINE
Visualisation de la ligne suivante du texte.



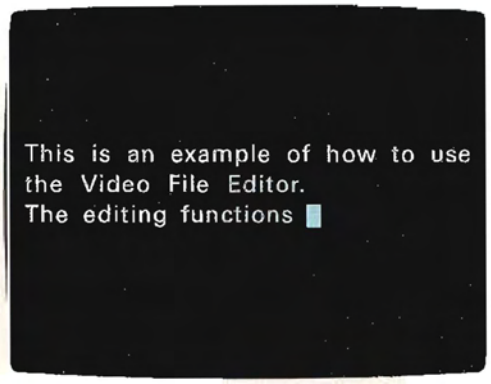
INSERT MODE
Passage en mode insertion pour ajouter le point et la majuscule.



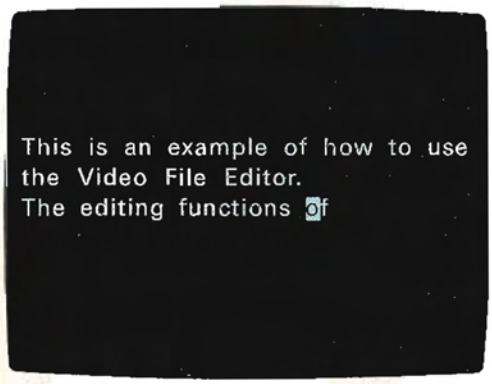
END OF LINE
Positionnement du curseur sur la colonne qui suit le dernier caractère non blanc de la ligne.



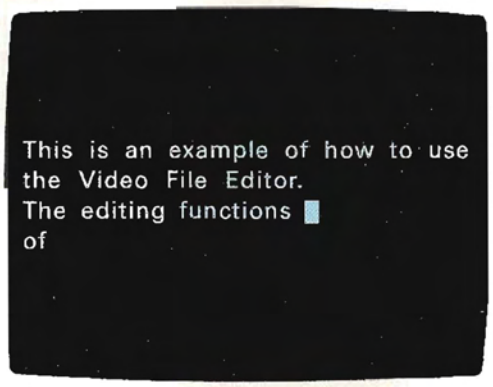
BACKSPACE
Déplacement du curseur d'une colonne vers la gauche et suppression du caractère ainsi pointé. Ici, la fonction a été utilisée à deux reprises.



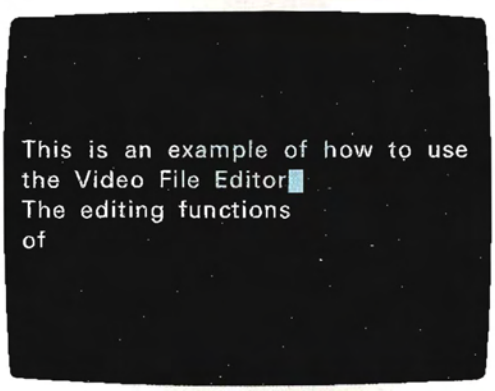
RECALL LINE
Restitution du précédent contenu de la ligne. Cependant, le curseur ne retrouve pas son ancienne position.



SPLIT LINE
Renvoi à la ligne suivante des caractères situés à droite du curseur (y compris celui sur lequel il se trouve).



CURSOR UP
Déplacement du curseur d'une ligne vers le haut.



INSERT LINE
Insertion d'une ligne « blanche » derrière celle où se trouve le curseur et positionnement de ce dernier au début de la nouvelle ligne.

```
This is an example of how to use
the Video File Editor.

The editing functions
of
```

Les deux exemples suivants ont trait à la recherche de chaînes de caractères. La recherche sera effectuée dans le texte reproduit ci-contre et portera d'abord sur la chaîne « func », puis sur la chaîne « e of ».

```
This is an example of how to use
the search function keys of the
Video File Editor to find a
particular combination of charac-
ters
```

Si l'on entre la commande SEARCH DOWN (CTRL+8) après avoir écrit « func » sur la ligne de commande, la recherche commence à l'endroit où se trouve le curseur et se poursuit vers la fin du texte.

```
func

This is an example of how to use
the search function keys of the
Video File Editor to find a
particular combination of charac-
ters
```

Si, au contraire, on entre SEARCH UP (COMMAND+8), la recherche de la chaîne (ici « e of ») va s'effectuer en sens inverse, en remontant de la position du curseur vers le début du texte.

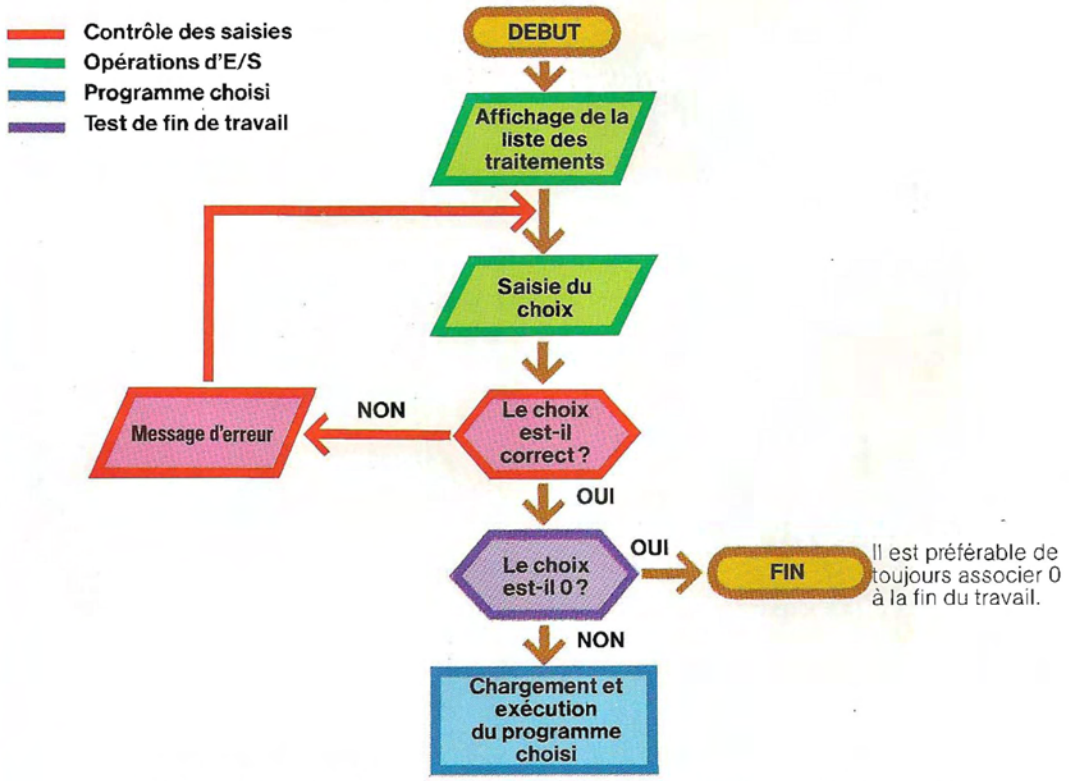
```
e of

This is an example of how to use
the search function keys of the
Video File Editor to find a
particular combination of charac-
ters
```

de modules indépendants, dont chacun forme un tout.
Certes, ces différentes parties Une fois ces modules définis, il y a deux façons de procéder : soit on subordonne leur exécution à un ordre de priorité fixé dans le programme (il s'agit alors de routines classiques), soit on les choisit au fur et à mesure des besoins. Dans ce dernier cas, les modules doivent être indépendants, puisque l'utilisateur lui-même établit librement l'ordre qui lui convient le mieux.
Si le traitement requiert le respect d'un enchaînement précis des tâches, l'exécution doit être pilotée par le système et l'on préférera généralement la première méthode.
De toute façon, un bon logiciel doit toujours prévoir

- de présenter à l'utilisateur la liste des différents traitements possibles ;
- de lui demander son choix et de le contrôler ;
- de lui exposer, sur sa demande, les caractéristiques et la logique générale des programmes qui constituent cette application.

Le programmeur a d'ailleurs tout à gagner en s'astreignant à cette discipline. Il doit structurer son logiciel : la rédaction s'en trouve facilitée et le découpage en plusieurs modules évite bien des problèmes (dépassement de capacité, par exemple).
Les trois points énumérés plus haut sont généralement traités dans un module de **menu**, qui offre à l'utilisateur la possibilité de travailler, en quelque sorte, « à la carte ».
Les menus ont une structure très simple : ils affichent la liste des différents traitements proposés, et demandent l'entrée du code associé au traitement désiré.
Des instructions d'appel commandent ensuite le chargement et l'exécution du module choisi.
Ci-dessous se trouve l'organigramme d'un menu type. Le programme choisi par l'utilisateur est chargé et prend la place du menu en mémoire centrale.

ORGANIGRAMME D'UN MENU TYPE



Tous les programmes ainsi appelés doivent se terminer par une instruction permettant de recharger le menu en mémoire, en vue de la sélection du traitement suivant.

En fin d'exécution, le programme appelé doit, à son tour, recharger le menu en mémoire afin de permettre un choix ultérieur. L'instruction réalisant ces chargements est la suivante :

RUN "X:NOM"

X désigne l'unité disque où réside le programme à appeler (A/B ou 0/1) et NOM le nom par lequel le programme à exécuter est identifié sur disque.
On peut concevoir plusieurs menus imbriqués, afin de multiplier les possibilités de choix et faciliter la recherche de la fonction désirée. Par exemple, un menu principal peut appeler un certain nombre de menus secondaires, chacun étant consacré à une certaine catégorie de fonctions ; à leur tour, les menus secondaires appellent des programmes spécifiques. A la fin de chaque programme, on devra recharger le menu secondaire à partir duquel il a été appelé, et ce dernier appellera le menu principal. De cette manière, le choix sera possible dans les deux sens : d'une notion générale, arriver au détail et, à l'opposé, de l'application particulière, remonter au cas général. Cette page et la suivante illustrent le listing d'un menu principal appelant d'autres menus. Ce programme constituant le point d'entrée de toute la procédure, on doit y prévoir les fonctions communes à tous les programmes, comme l'initialisation des touches de fonction ou la définition d'une forme particulière du curseur.
On remarquera, à cette occasion, que les touches de fonction sont seulement initialisées, c'est-à-dire qu'un code numérique leur est affecté. La fonction à exécuter sous ce code sera, elle, établie par chaque programme. Par exemple, dans le menu principal, la valeur numérique 2 peut être attribuée à une touche donnée : ce code, dans un programme déterminé, pourra faire débuter les fonctions d'impression, dans un autre programme il pourra

MENU PRINCIPAL

```
4540 ' PROGRAMME MENU
4541 ' *****FICHIERS*****
4542 '
4543 '    NOM       DRV    DESCRIPTION                OCTETS     CHAMPS
4544 '
4545 ' REPCLI         A     Repertoire clients          128         7
4546 ' REPFOU         A     Repertoire fournisseurs     128         7
4547 ' MARGE H        A     Marge hebdomadaire           91         9
4548 ' MARGE G        A     Marge generale              100        10
4549 ' FACTUR         B     Factures                     70         9
4550 ' RECETT         B     Recettes generales           70         9
4551 ' PAIES          B     Paiements generaux           70         9
4552 ' NCRED          B     Notes de credit              70         9
4553 ' TTVA           A     Taux TVA                    128         7
4554 ' TCRED          A     Taux credit                 128         7
4555 '
4556 LPRINT CHR$(27)+CHR$(62)    ' definition de l'espacement
4560 GOSUB 6040          ' Initialisation des touches de fonction
4580 BI$=CHR$(27)+"+"+CHR$(7)
4600 GOSUB 4960
4620 ON V GOTO 4640, 4660, 4680, 4700, 4720, 4740, 4760, 4780, 4800, 4820
4640 RUN "A:MENU1"          ' STOCK
4660 RUN "A:MENU2"          ' REPERTOIRE CLIENTS ET FOURNISSEURS
4680 RUN "A:MENU3"          ' RECETTES
4700 RUN "A:MENU4"          ' FACTURATION
4720 RUN "A:MENU5"          ' MARCHANDISES
4740 RUN "A:MENU6"          ' NOTES DE CREDIT
4760 RUN "A:RECLI"          ' RECUS CLIENTS
4780 RUN "A:REFOUR"         ' RECUS FOURNISSEURS
4800 RUN "A:HLP1"
4820 PRINT BI$
4840 X=30 : Y=10 : GOSUB 5700
4860 PRINT CHR$(27)+"G1"+"FIN DE TRAVAIL"     ' message souligne
4880 PRINT CHR$(27)+"G0"                      ' video normale
4900 STOP
4920 END
```

```
4940          ***** AFFICHAGE DU MENU *****
4960 PRINT BI$
4980 X=2 : Y=1 : GOSUB 5700
4990 PRINT "COMPTABILITE GENERALE"
5000 X=15 : Y=6 : GOSUB 5700
5020 PRINT " 1 - STOCK "
5040 Y=Y+1 : GOSUB 5700
5060 PRINT " 2 - REPERTOIRE CLIENTS ET FOURNISSEURS "
5080 Y=Y+1 : GOSUB 5700
5100 PRINT " 3 - RECETTES ET PAIEMENTS "
5120 Y=Y+1 : GOSUB 5700
5140 PRINT " 4 - FACTURATION "
5160 Y=Y+1 : GOSUB 5700
5180 PRINT " 5 - MARCHANDISES "
5200 Y=Y+1 : GOSUB 5700
5220 PRINT " 6 - NOTES DE CREDIT "
5240 Y=Y+1 : GOSUB 5700
5260 PRINT " 7 - RECUS CLIENTS "
5280 Y=Y+1 : GOSUB 5700
5300 PRINT " 8 - RECUS FOURNISSEURS "
5320 Y=Y+1 : GOSUB 5700
5340 PRINT " 9 - Explications sur l'utilisation du programme "
5360 Y=Y+1 : GOSUB 5700
5380 PRINT " 0 - FIN DE TRAVAIL "
5400 X=1 : Y=Y+2
5410 GOSUB 5700
5420 PRINT " ENTRER L'OPTION CHOISIE : "
5440 X=26 : GOSUB 5700
5450 A$=INPUT$(1)
5460 PRINT A$
5470 U=VAL(A$)
5480 IF U>9 GOTO 5600        ' ERREUR DE SAISIE
5500 PRINT BI$
5520 IF U=0 THEN U=10 : GOTO 5680        ' SORTIE
5540 X=5 : Y=5 : GOSUB 5700
5560 PRINT "PATIENCE ! On charge le programme"
5580 GOTO 5680
5600 PRINT CHR$(7), CHR$(7)        ' TRAITEMENT D'ERREUR
5620 X=42 : GOSUB 5700
5640 PRINT "CODE NON VALIDE"
5660 GOTO 5440
5680 RETURN
5685 '
5690 '
5695 '
5700 ' **** POSITIONNEMENT DU CURSEUR ****
5720 '       EN COORDONNEES PARAMETREES (X,Y)
5740 PRINT CHR$(27)+CHR$(61)+CHR$(31+Y)+CHR$(31+X) ;
5760 RETURN
5780 ' On ajoute 31 aux coordonnees pour obtenir la
5800 ' position en hexadecimal
5820 '
5840 ' **** INITIALISATION DES TOUCHES DE FONCTION *****
5860 PRINT CHR$(27)+"c4AA"        ' Forme du curseur
5880 A$=CHR$(27)+CHR$(46)        ' Partie commune de la chaine
5900 '
5920 B=98 : I1=49 : I2=48
5940 FOR I = 1 TO 8
5960 T$=A$+CHR$(B)+CHR$(I1)+CHR$(I2)
5980 PRINT T$;          ' Affectation aux touches B de 1 a 8 des
6000 B=B+1              ' par la variable B)
6020 I2=I2+1            ' Les touches vont donc de 98 en hexa
6040 NEXT I             ' par la chaine I1 I2 (de 10 a 17)
6060 B=106 : I1=48 : I2=52
6080 FOR I=1 TO 4
6100 T$=A$+CHR$(B)+CHR$(I1)+CHR$(I2)
6120 PRINT T$;          ' On affecte aux touches j a m
6140 B=B+1              ' les fonctions codees en hexa de 04 a 07
6160 I2=I2+1
6180 NEXT I
6200 RETURN
```

servir de clé de sortie du traitement, etc. Notre listing contient les instructions nécessaires à la gestion du curseur et des touches fonction d'une machine bien déterminée ; si le menu devait être utilisé sur un système différent, ces instructions devraient être modifiées (comme on l'a déjà dit, les codes utilisables sont spécifiques à chaque machine).
Pour faciliter l'adaptation du logiciel présenté (on parle alors de « portabilité »), les fonctions spécifiques étroitement liées au matériel hardware sont développées séparément, dans des routines appropriées.
Si l'on se réfère toujours au listing de la page 625 (menu principal), on remarque une partie initiale de commentaires, qui énumèrent les fichiers utilisés dans la procédure (lignes 4545 à 4555) : le nom de chaque fichier, l'unité disque (driver) où doit être chargée la disquette, le contenant (colonne DRV), la description du contenu, la longueur des enregistrements en octets et le nombre de champs constituant les enregistrements. Cette description initiale des fichiers de données nécessaires au programme se révélera très utile quand des modifications seront nécessaires.
La ligne 4560 appelle le sous-programme 6040, qui définit la forme du curseur et initialise les touches de fonction.
La chaîne BI$ (ligne 4580) contient les caractères ESCAPE [CHR$(27)], + et CHR$(7). ESCAPE provoque le passage en mode commande. Le deuxième caractère pour l'unité d'écran considérée commande l'effacement de l'écran. Le dernier émet un son afin d'appeler l'attention de l'opérateur ; il s'agit d'un « bip » qui, sur certaines machines, est généré par l'instruction spéciale BEEP. On remarquera que le code CHR$(7) permettant d'obtenir le bip est conforme aux tables ASCII (CTRL + G) et est donc présent sur presque toutes les consoles.
La présentation des différentes fonctions disponibles (affichage du menu) s'effectue dans le sous-programme 4960, qui commence par initialiser l'écran (PRINT BI$). La position sur l'écran des différentes lignes est paramétrée par le sous-programme 5700. En effet, la ligne 4980 comporte deux paramètres (X et Y) qui seront transmis à la routine. Les valeurs de X et Y indiquent respectivement la colonne et la ligne où doit commencer l'impression.

Le micro-ordinateur simplifie considérablement la gestion d'un stock.

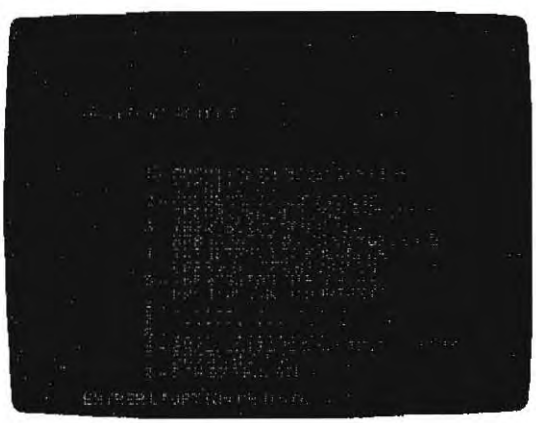
Olivetti

MENU DU STOCK

```
4500 '
4505 ' ** MENU DU STOCK **
4510 '    PROGRAMME MENU1
4512 '
4514 '
4516 ' Les touches fonction sont initialisees dans le programme MENU
4518 '
4520 BI$=CHR$(27)+"+"+CHR$(7)
4525 GOSUB 4630
4526 '
4530 ON V GOTO 4535, 4540, 4545, 4550, 4555, 4560, 4565, 4570, 4575, 4580
4535 RUN "A:CREART"          ' Creation d'articles
4540 RUN "A:MOUHEB"          ' Mouvements hebdomadaires
4545 RUN "A:PRMOUH"          ' Impression mouv. hebdo.
4550 RUN "A:TRANSF"          ' Transfert vers fichier archives
4555 RUN "A:IMPRES"          ' Impression fichier archives
4560 GOTO 4510               ' Ces 3 lignes sont
4565 GOTO 4510               ' prevues pour l'elargissement
4570 GOTO 4510               ' ulterieur de MENU1
4575 RUN "A:HLP2"
4576 '
4580 PRINT BI$
4585 X=30 : Y=10
4590 GOSUB 4870              ' Deplacement du curseur
4595 PRINT CHR$(27)+"G1"+"fin de travail"    ' message souligne
4600 PRINT CHR$(27)+"G0"                     ' video normale
4605 RUN "A:MENU"
4610 END
4615 '
4620 '
4625 ' ** AFFICHAGE DU MENU **
4630 '
4635 PRINT BI$
4640 X=2 : Y=1 : GOSUB 4870
4645 PRINT "Gestion du stock"
4650 X=15 : Y=6 : GOSUB 4870
4655 PRINT "1 - Creation et modification d'articles"
4660 Y=Y+1 : GOSUB 4870
4665 PRINT "2 - Mouvements chargement/dechargement hebdomadaires"
4670 Y=Y+1 : GOSUB 4870
4675 PRINT "3 - Impression mouvements hebdomadaires avec montants"
4680 Y=Y+1 : GOSUB 4870
4685 PRINT "4 - Transfert des mouvements vers fichier archives"
4690 Y=Y+1 : GOSUB 4870
4695 PRINT "5 - Impression du fichier archives avec montants"
4700 Y=Y+1 : GOSUB 4870
4705 PRINT "6 - ................"
4710 Y=Y+1 : GOSUB 4870
4715 PRINT "7 - ................"
4720 Y=Y+1 : GOSUB 4870
4725 PRINT "8 - ................"
4730 Y=Y+1 : GOSUB 4870
4735 PRINT "9 - Explications sur l'utilisation du programme"
4740 Y=Y+1 : GOSUB 4870
4745 PRINT "0 - Fin de travail"
4750 Y=1 : Y=Y+2
4755 GOSUB 4870
4760 PRINT "ENTRER L'OPTION CHOISIE:"
4770 A$=INPUT$(1)
4775 PRINT A$
4780 V=VAL(A$)
4785 IF V>9 GOTO 4820        ' ERREUR
4790 PRINT BI$
4795 IF V=0 THEN V=10 : GOTO 4840        ' Sortie
4800 X=5 : Y=6
4805 GOSUB 4870
4810 PRINT "PATIENCE ! On charge le programme"
4815 GOTO 4840
4820 PRINT CHR$(7), CHR$(7)        ' Traitement d'erreur
4825 X=42 : GOSUB 4870
```

```
4830 PRINT "CODE NON VALIDE"
4835 GOTO 4765
4840 RETURN
4845 '
4850 '
4855 ' ** POSITIONNEMENT DU CURSEUR **
4860 '    en coordonnees (X,Y)
4870 PRINT CHR$(27)+CHR$(61)+CHR$(31+Y)+CHR$(31+X);
4875 RETURN
```

Menu principal : selon l'option choisie, l'un des menus secondaires est chargé et présenté.



On a choisi l'option 1 (magasin). La sélection suivante permet l'accès aux différentes procédures.

Pour préparer l'écriture à partir des coordonnées X, Y (colonne, ligne), on commencera donc par un appel au sous-programme 5700 de déplacement du curseur ; l'écriture de la chaîne d'en-tête s'effectuera ensuite par une instruction PRINT classique. Le déplacement du curseur se programme simplement en envoyant à l'écran la chaîne de commande (ligne 5740), terminée par le symbole ; qui empêche le retour à la ligne.

A la ligne 5450 figure l'instruction A$= INPUT$(1), permettant la saisie d'un unique caractère (le choix peut se faire seulement entre 0 et 9) ; le caractère ainsi acquis est transformé en numérique (ligne 5470) et contrôlé (ligne 5480). Si la valeur est correcte, on efface l'écran (ligne 5500), un message s'affiche (ligne 5560) et le contrôle est rendu au programme principal (instruction 4620).

La sélection du programme à charger est effectuée en fonction du choix de l'opérateur. Si, par exemple, la réponse est 1 (V=1), le programme MENU1 est chargé et exécuté.

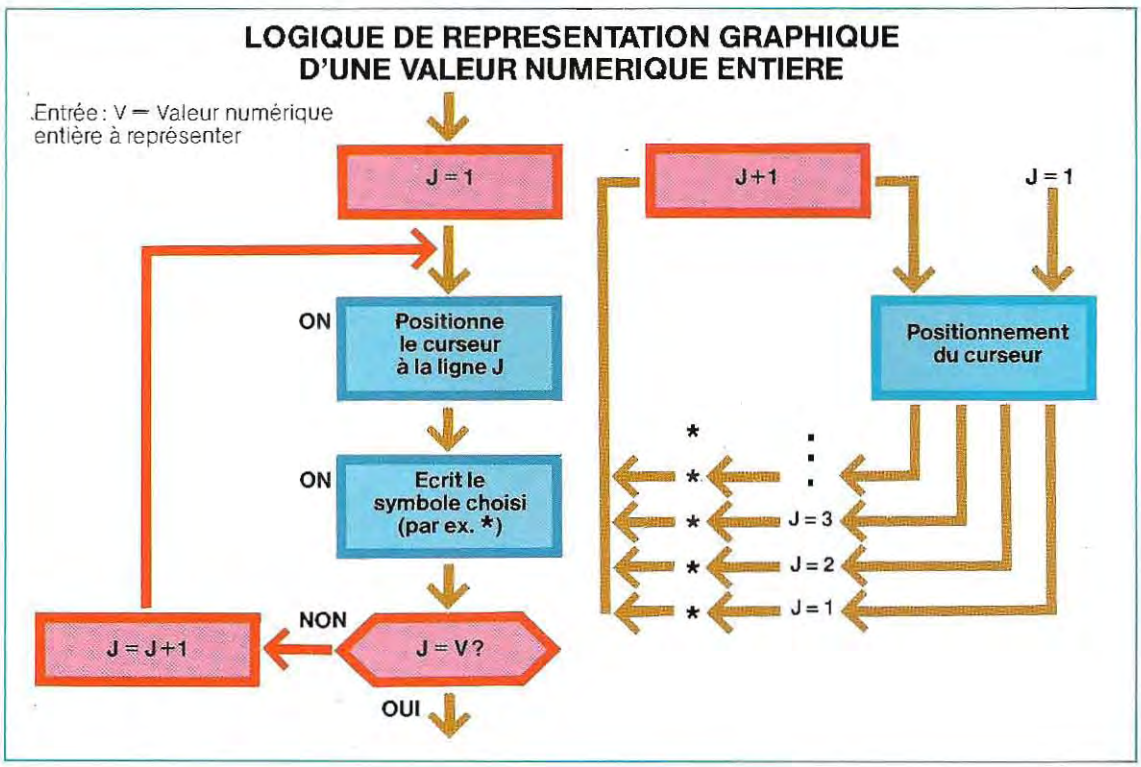
Ce programme (dont le listing est illustré sur la page 628 et ci-dessus) permet d'effectuer de nouveaux choix (lignes 4535 à 4575) et, à son terme, rappelle le menu précédent (ligne 4605) : il s'agit donc d'un menu secondaire. Il ne figure pas ici d'instructions pour obtenir une forme particulière du curseur, ni d'initialisation des touches fonction (opérations déjà exécutées dans le menu principal) ; par contre, le sous-programme de déplacement du curseur (ligne 4870) doit toujours être présent. On remarque que les deux menus (principal et secondaire) possèdent la même numérotation : l'un recouvre l'autre et il n'est pas nécessaire d'utiliser des numérotations différentes. Au contraire, en partant du même numéro de ligne, la recherche des erreurs est facilitée ainsi que les modifications apportées aux programmes.

Génération d'histogrammes

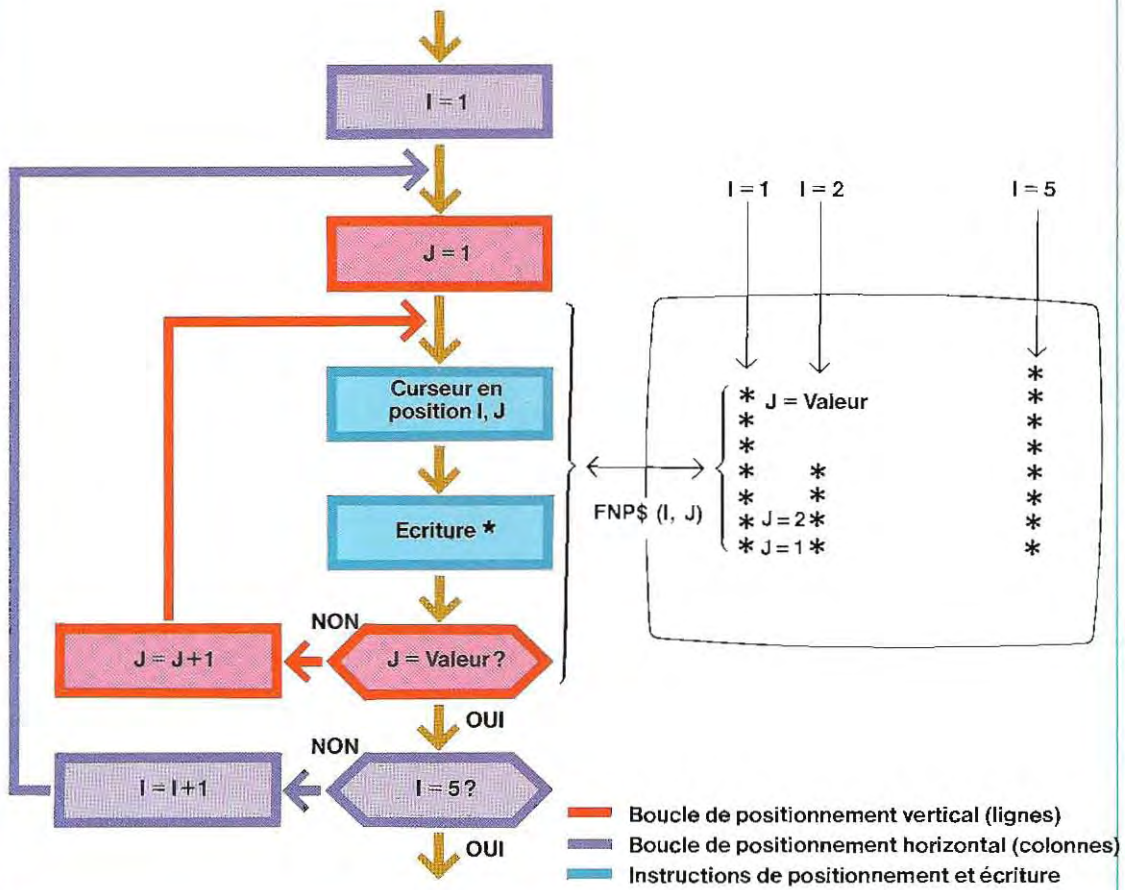
En utilisant la fonction d'adressage du curseur, il est possible de dessiner des histogrammes, même sur des écrans uniquement alphanumériques.
La procédure consiste à représenter sur l'écran une série de symboles, en nombre proportionnel (selon un facteur convenu) à la grandeur à représenter. Par exemple, en choisissant un facteur 10, la valeur 150 sera représentée par une colonne de 15 symboles (150/10).
Pour l'afficher à l'écran, il faudra écrire 15 fois le même symbole, en décrémentant chaque fois d'une unité le numéro de la ligne, tout en gardant inchangé le numéro de colonne. De cette manière, on « dessine » une petite colonne de hauteur proportionnelle à la valeur numérique à représenter.
Le schéma ci-dessous illustre le principe de cette logique de représentation.
La position du curseur peut se calculer en définissant une fonction contenant les codes d'adresage. Par exemple, la fonction :

```
FNP$(A,B)=CHR$(27)+CHR$(61)
          +CHR$(31+B)+CHR$(31+A)
```

positionne le curseur à la ligne B, colonne A. En effet, en ajoutant 31 (code hexa de 1) à la valeur décimale A (ou B), on obtient la valeur hexadécimale exacte de la position du curseur.
Comme l'écran alphanumérique possède généralement 80 colonnes et 24 lignes, les valeurs maximales envisageables sont A=80 et B=24, alors que le déplacement minimal sera pour l'axe X (indiqué par I) d'un caractère et pour l'axe Y (indiqué par J) d'une ligne. En d'autres termes, on ne peut afficher des symboles plus rapprochés que la distance existant entre deux lignes (axe Y) ou entre deux caractères (axe X) : sur un écran alphanumérique (non graphique) la possibilité d'adressage se limite à un nombre entier de lignes et de colonnes (caractères) : les positions intermédiaires n'existent pas.
Sur la page 631, la première colonne de l'histogramme, représentant la valeur 150, sera obtenue en positionnant le curseur sur 15 lignes successives (1, 2, 3, etc.) et en écrivant le symbole choisi, la valeur de la colonne (1) restant inchangée. Pour représenter, à côté de la précédente, une nouvelle valeur, il faut changer l'abscisse (colonne I) et répéter la boucle sur les lignes (J). En réunissant les



REPRESENTATION GRAPHIQUE SUR PLUSIEURS COLONNES



deux boucles, c'est-à-dire en les imbriquant l'une dans l'autre, on peut obtenir des histogrammes comprenant jusqu'à 80 colonnes disposées côte à côte.

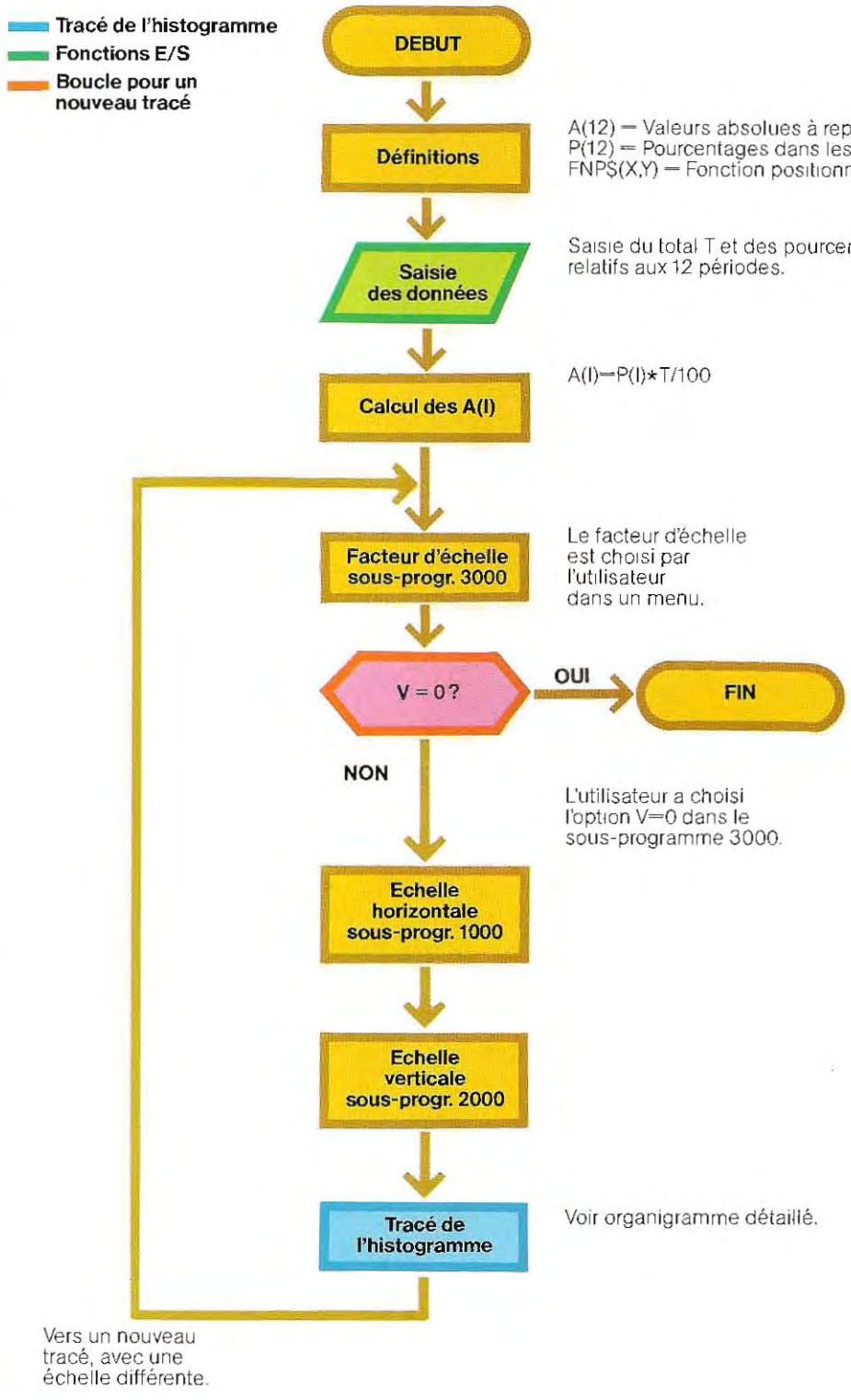
Le schéma de cette page illustre la préparation d'un histogramme représentant consécutivement cinq valeurs numériques. La première boucle, la plus externe, sélectionne la colonne de l'écran sur laquelle se fera la représentation graphique de la valeur. La boucle la plus interne détermine la hauteur de chacune des cinq petites colonnes, en répétant sur un nombre de lignes proportionnel à la valeur qui doit être représentée le symbole choisi (dans l'exemple le symbole *).

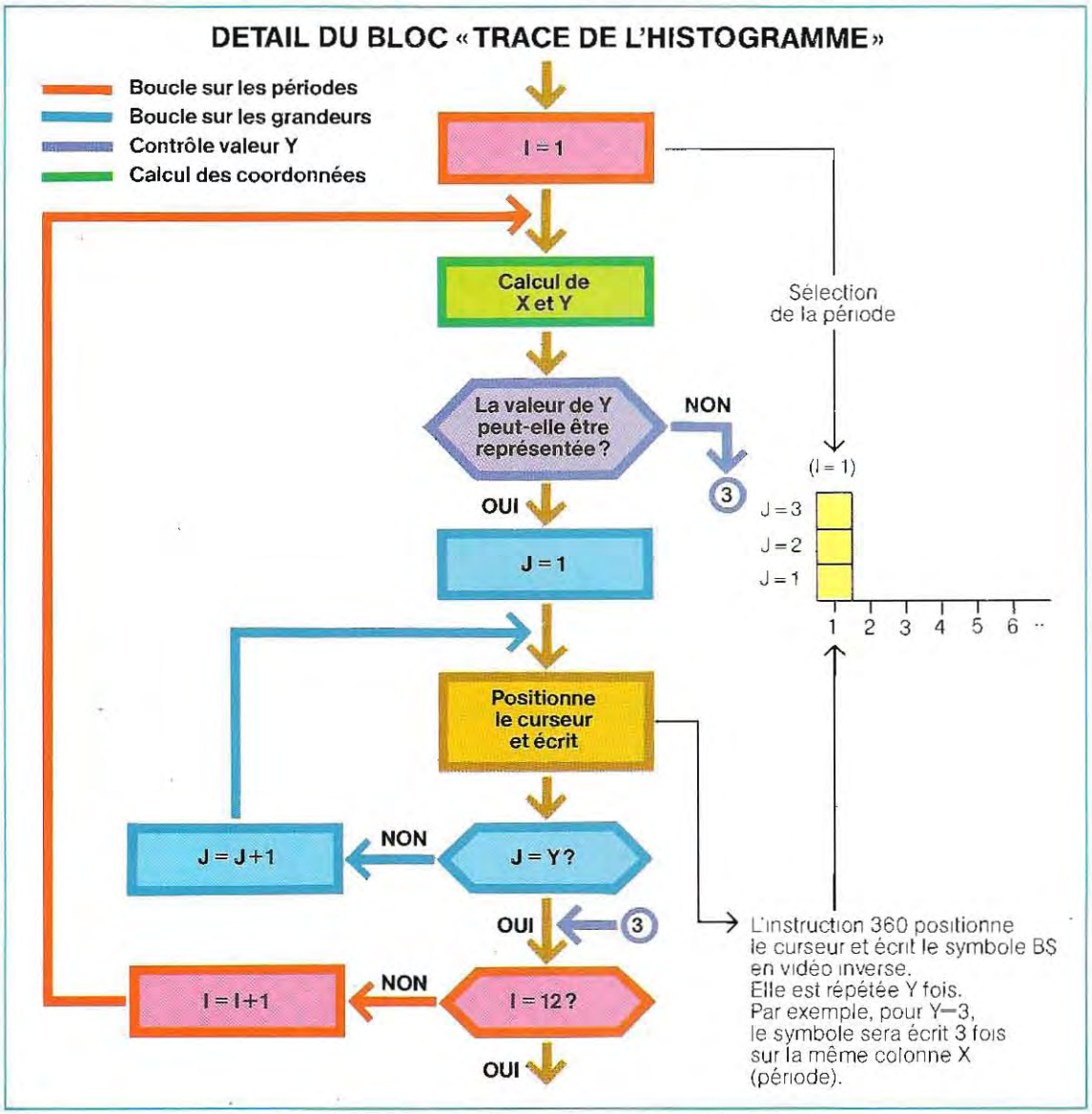
Sur le schéma, les coordonnées X et Y du curseur sont désignées par les symboles I et J, pour distinguer ce cas particulier d'adressage discret du cas général d'adressage par coordonnées.

Sur les écrans alphanumériques classiques, les coordonnées du curseur peuvent prendre comme valeurs des nombres entiers compris entre 1 et 24 (pour X), et entre 1 et 80 (pour Y), représentant effectivement à l'écran des lignes et des colonnes de caractères. En d'autres termes, on ne peut pas positionner le curseur au point 2.75 (lignes) : ou bien on choisit la position (ligne) 2 ou bien la position 3. En ce cas, on dit que l'adressage est discret (limité à un certain nombre de valeurs).

Sur les écrans graphiques, si les coordonnées restent entières, on n'est cependant plus lié aux lignes et aux colonnes ; on peut adresser les points de l'écran, caractérisés par un couple de coordonnées, comme les points d'une feuille de papier millimétré. Même dans ce cas, toutes les valeurs possibles de X et Y ne peuvent être représentées, car les points de l'écran, même si ce dernier est très grand,

PROGRAMME DE GENERATION D'HISTOGRAMMES





sont en nombre limité.

Les consoles graphiques professionnelles possèdent un écran à « haute résolution », dont la définition atteint, pour les modèles haut de gamme en noir et blanc, 4096 × 4096 points (ou pixels) adressables. Certaines consoles ouvrent des perspectives entièrement nouvelles grâce à l'apport de la couleur : si la définition de l'écran s'abaisse alors à 1024 × 1024, l'utilisateur peut afficher simultanément jusqu'à 4096 couleurs, choisies dans une palette de base de 4 millions de nuances.

Ces deux pages illustrent l'organigramme d'un programme de préparation d'histogrammes de 12 valeurs numériques. Cet exemple, de portée générale, peut s'appliquer à des cas pratiques de représentation graphique de la distribution d'une grandeur quelconque en 12 périodes. Il sera utilisé, par exemple, pour représenter des budgets, des objectifs de travail, etc. ; le nombre des périodes (12) permet l'écriture d'une représentation mensuelle sur l'étendue d'une année.

Les données requises par le programme en entrée (instructions 100 à 150) concernent la valeur totale (montant de la dépense, heures de travail, etc.) et le pourcentage de ce total prévu pour chacune des périodes.

Le programme calcule la valeur absolue (A(I))

EXEMPLE DE PREPARATION D'HISTOGRAMMES

```
30 ' PROGRAMME HISTO
40 DIM A(12),P(12)
50 DEF FNP$(X,Y)=CHR$(27)+CHR$(61)+CHR$(31+Y)+CHR$(31+X)
60 C$=CHR$(27)+"+"              ' pour effacer l'ecran
70 B$=""                        ' motif composant l'histogramme
80 FI$=CHR$(27)+"G4"            ' passage en video inverse
90 FN$=CHR$(27)+"G0"            ' retour en video normale
100 INPUT "MONTANT TOTAL ";T
110 PRINT "ENTREZ LE POURCENTAGE DE CHAQUE PERIODE"
120 FOR I=1 TO 12
130 PRINT "Periode ";I
140 INPUT " Pourcentage: ";P(I)
150 NEXT I
160 EH=5        ' Facteur d'echelle horizontale (1 periode = 5 positions)
170 ' ** Calcul de la valeur de chaque periode **
180 FOR I=1 TO 12
190 A(I)=P(I)*T/100
200 NEXT I
205 PRINT C$            ' efface l'ecran
210 '
220 GOSUB 3000          ' choix de l'echelle verticale
230 IF V=0 GOTO 440     ' sortie du programme
240 '
250 EV=20/MX            ' calcul du facteur d'echelle verticale
255 '
260 GOSUB 1000            ' trace echelle horizontale
270 GOSUB 2000            ' trace echelle verticale
275 '
280 ' ** Boucle de trace de l'histogramme
290 FOR I=1 TO 12
300 X=I*EH
310 Y=A(I)*EV
320 IF Y<1 GOTO 390       ' Valeur non representable
330 FOR J=1 TO Y
340 L=22-J              ' La position J=1 se trouve en haut a gauche.
345 '                   Pour partir du bas, on se place en 22-J (la ligne
350 '                   22 est occupee par l'echelle horizontale)
360 PRINT FNP$(X,L);FI$;B$;FN$
370 NEXT J
380 X1=X-2 : PRINT FNP$(X1,L);P(I)
390 NEXT I
400 X=40 : Y=1 : PRINT FNP$(X,Y);
410 PRINT "Pour continuer, taper un caractere ":G$=INPUT$(1)
420 PRINT C$
425 '
430 GOTO 220
440 END
1000 ' ** Ecriture de l'echelle horizontale (12 periodes)
1010 NL=EH*12+1             ' Longueur de l'echelle
1020 G$=STRING$(NL,"-")     ' Une ligne de - symbolise l'echelle horizontale
1030 X=1 : Y=22 : PRINT FNP$(X,Y); G$
1040 Y=23
1050 FOR I=1 TO 12
1060 X=I*EH
1070 PRINT FNP$(X,Y); I;
1080 NEXT I
1090 RETURN
2000 ' ** Ecriture de l'echelle verticale
2010 X=1
2020 FOR I=1 TO 20          ' 20 = hauteur de l'ecran
2030 L=22-I
2040 PRINT FNP$(X,L); "!"       ' Une ligne de ! symbolise l'echelle verticale
2050 NEXT I
2060 PRINT FNP$(X,1); "Total = ";T
2070 RETURN
3000 ' ** Choix de l'echelle verticale **
3010 X=2 : Y=1 : PRINT FNP$(X,Y);
3020 PRINT "CHOIX DE L'ECHELLE VERTICALE"
3030 X=15 : Y=6 : PRINT FNP$(X,Y);
```

```
3040 PRINT "1 - relative a la valeur maximale des pourcentages"
3050 Y=Y+1 : PRINT FNP$(X,Y);
3060 PRINT "2 - relative a la valeur totale"
3070 Y=Y+1 : PRINT FNP$(X,Y);
3080 PRINT "3 - autre echelle"
3090 Y=Y+1 : PRINT FNP$(X,Y);
3100 PRINT "4 - fin de travail"
3110 X=1 : Y=Y+2 : PRINT FNP$(X,Y);
3120 PRINT "ENTREZ L'OPTION CHOISIE";
3130 A$=INPUT$(1) : PRINT A$
3140 V=VAL(A$)
3150 ON V GOTO 3155, 3210, 3220, 3230
3155 ' * V=1 : valeur relative au pourcentage maximal *
3160 MX=0
3170 FOR I=1 TO 12
3180 IF A(I)>MX THEN MX=A(I)
3190 NEXT I
3200 GOTO 3230
3210 MX=T : GOTO 3230    ' * V=2 : valeur relative au total T
3220 INPUT "Valeur d'echelle choisie"; MX    ' * V=3 : valeur choisie
3230 RETURN
```

correspondant à chaque période (lignes 170 à 200), et fixe l'échelle horizontale à 5 (EH) : les colonnes de l'histogramme seront espacées de cinq unités.

Trois sous-programmes sont prévus. La routine 3000 met en évidence l'effet du changement d'échelle, en proposant à l'utilisateur trois types de représentation. Dans le premier cas, la hauteur maximale (vingt lignes en Y) est utilisée pour représenter la valeur maximale des A(I). Dans le second cas, c'est la valeur totale (T) qui est dessinée sur vingt lignes. Les valeurs seront ensuite représentées par un pourcentage de la hauteur de la colonne échelle.

Remarquons que cette routine 3000 représente un nouvel exemple de menu, qui permet la visualisation successive de différents modes de représentation.

Les lignes 400 et 410 constituent un exemple type de message suivi d'un temps d'attente (temporisation). Pour continuer le traitement qui suit l'affichage, il faut attendre que l'utilisateur ait lu les résultats montrés à l'écran. On réserve généralement deux ou trois lignes au dialogue avec l'utilisateur, et l'on fait figurer sur une de ces lignes le message choisi (ligne 410). L'attente se réalise par l'instruction S$= INPUT$(1) qui interrompt l'exécution du programme, jusqu'à l'entrée d'un caractère quelconque. Comme l'instruction supprime l'écho, le caractère n'apparaît pas à l'écran et la chaîne S$ ne sert qu'à réaliser l'attente.

Notre programme utilise comme symbole pour le tracé des histogrammes un espace, affiché en vidéo inverse (il se présente sous la forme d'un rectangle lumineux) ; le symbole est affecté à la variable B$ (ligne 70) et pour le remplacer, il suffit de changer l'affectation.

On définit la position du curseur et l'écriture du symbole par l'instruction 360. La fonction FNP$(X,L) positionne le curseur à la colonne X, ligne L ; le numéro de ligne est déterminé par l'expression 22-Y, de façon à tracer l'histogramme à partir de la ligne 22, dans la partie inférieure de l'écran. La variable FI$ contient les codes de commande pour passer en vidéo inverse (ligne 80) ; B$ est le symbole à écrire et FN$ contient les codes chargés de remettre l'écran en mode d'affichage normal (ligne 90). A l'extrémité d'une colonne représentant une valeur donnée, on écrit le pourcentage P(I) correspondant (ligne 380). On obtient l'écriture de cette valeur en laissant le curseur positionné sur la ligne L de sortie de la boucle, mais déplacé de deux colonnes à gauche (X1=X–2) afin d'aligner le nombre avec le sommet de la colonne.

Les codes employés ici se réfèrent à une machine particulière : pour d'autres modèles, il faut se conformer aux valeurs spécifiées dans les manuels d'utilisation.

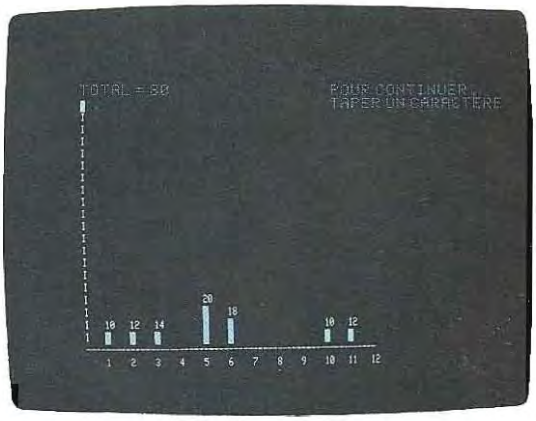
Les sous-programmes 1000 et 2000 tracent les échelles horizontales et verticales. En ligne 1020, on a utilisé l'instruction STRING$(NL,"–"), qui génère la chaîne G$ de longueur NL, formée de symboles – concaténés ; l'écriture de cette chaîne donne une ligne horizontale pointillée. De plus, sur l'échelle horizontale sont reportées les valeurs des intervalles (de 1 à 12).

L'échelle verticale est tracée à l'aide d'une boucle (lignes 2020-2050) qui affiche le caractère « I ».

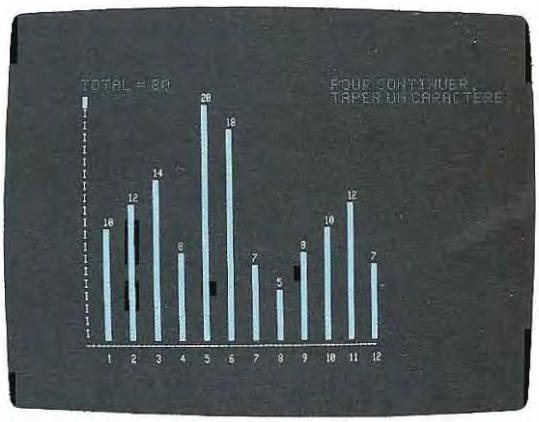
GENERATION D'HISTOGRAMMES A L'ECRAN

Sur cette page sont illustrés les écrans de la phase de saisie de données (ci-contre) et d'affichage des histogrammes (au-dessous), avec référence au programme illustré à la page 634. La phase de saisie est guidée par le programme lui-même, qui demande tour à tour les données nécessaires (lignes 100 à 150). Après la phase de calcul, le contrôle passe au sous-programme 3000 qui détermine l'échelle verticale à adopter.

Dans le premier cas, on a choisi l'échelle relative à la valeur totale. Celle-ci occupe donc une hauteur de 20 lignes.
Chaque donnée numérique est représentée par une colonne de hauteur égale à un pourcentage (c'est la donnée elle-même) de la hauteur maximale disponible.
Notons que la différence entre les valeurs 10, 12 et 14, par exemple, n'est pas représentable, et que la période 12 doit avoir une valeur inférieure à 5.



Dans le deuxième cas, on a choisi l'échelle relative au pourcentage le plus fort, cette fois, la valeur maximale saisie (20) est représentée sur toute la hauteur disponible.
Les autres données sont représentées proportionnellement.
On obtient alors un étalement optimal des données sur l'écran.



Test 18

1 / Qu'appelle-t-on attributs d'un caractère ?

2 / Parmi ces instructions, lesquelles sont erronées ?

```
10 PRINT A,B,C
20 FOR I=1 TO 20 : PRINT "ESSAI" ; : NEXT I
30 PRINT A,TAB(60),B,TAB(100),C
```

3 / Quel est le résultat des instructions suivantes ?

```
10 FOR I=1 TO 30
20 PRINT I,"Données",A$
30 NEXT I
```

4 / En supposant que la fonction FNP$(X,Y) positionne le curseur à la colonne X et la ligne Y, quelles sont les instructions qui permettent d'écrire une chaîne quelconque de caractères en un point quelconque de l'écran ? Les coordonnées (ligne, colonne) et la chaîne seront saisies au clavier et le programme se terminera en lisant colonne = 0 ou bien ligne = 0.

5 / En affectant à la chaîne INV$ les codes de génération de caractères en vidéo inverse et à NOR$ les codes de retour à l'affichage normal, comme doit-on modifier l'instruction de la question précédente pour obtenir l'édition de la chaîne en affichage inversé ?

Les solutions du test se trouvent en pages 644 et 645.

Gestion des fichiers

Les fichiers reconnus par le Basic sont de deux types : séquentiels et en accès direct. Dans les fichiers séquentiels, les données sont écrites (et lues) les unes après les autres ; en d'autres termes, on ne peut accéder à une donnée sans avoir d'abord balayé toutes celles qui la précèdent. Ces types de fichiers constituent généralement le support de données entrées au clavier.
Les fichiers en accès direct (ou « Random »), par contre, permettent la lecture et l'écriture de données en adressant directement l'enregistrement intéressé.
La logique d'accès aux données d'un fichier est la même dans les deux cas et comprend la série suivante de fonctions :

- ouverture du fichier ;
- lecture et/ou écriture des données ;
- fermeture du fichier (à la fin de son utilisation).

Chaque fonction possède une syntaxe différente selon qu'elle s'adresse à un fichier séquentiel ou en accès direct. Elles seront donc examinées par la suite séparément avec une référence aux deux cas mentionnés ci-dessus.

Fonctions d'accès aux fichiers séquentiels

Un fichier séquentiel peut être ouvert de deux manières : en sortie (écriture par le programme), indiqué par le symbole O (Output) ou en entrée (lecture par le programme), indiqué par le symbole I (Input). Le type d'accès (I/O) sera spécifié dans l'instruction d'ouverture, qui doit être exécutée avant tout accès aux données.
Cette instruction est :

OPEN"TYPE",N,"NOM"

où
TYPE prend une des deux valeurs O ou I.
N représente une valeur numérique, appelée "numéro d'unité logique", que l'instruction d'ouverture associe au fichier.
Par la suite, toutes les opérations d'accès aux données se réfèreront à ce numéro pour désigner le fichier.
NOM est le nom avec lequel le fichier a été créé.

Par exemple, l'instruction

OPEN"I",2,"A:ESSAI"

ouvre le fichier nommé ESSAI, résidant sur la première unité disque (A:), en mode lecture (I), et lui associe le numéro d'unité logique 2. Dans les opérations de lecture de données, ce fichier sera ensuite désigné par le numéro 2. Sous certains systèmes, le numéro du fichier doit être précédé du symbole #. En ce cas, on modifie l'instruction d'ouverture de la manière suivante :

OPEN"I",#2,"A:ESSAI"

La modalité d'ouverture du fichier dépend des fonctions que le programme doit exécuter. Pour acquérir des données à parti du fichier, il faut l'ouvrir en mode INPUT : en effet, il s'agit d'une saisie de données par le programme. A l'opposé, pour écrire dans un fichier, on l'ouvrira en mode OUTPUT, car le fichier, pour le programme, est équivalent à un dispositif de sortie.
Après avoir accédé aux données, le fichier doit être fermé. L'instruction à utiliser est

CLOSE N

N étant la valeur numérique associée au fichier en phase d'ouverture. Par exemple, pour fermer le fichier précédent, on écrit l'instruction CLOSE 2.
Entre les instructions OPEN et CLOSE, on peut exécuter tooutes sortes de fonctions d'accès aux données ou de contrôle de l'état du fichier.

Voici la liste des instructions Basic opérant sur des fichiers séquentiels :

INPUT, LINE INPUT, WRITE, PRINT, PRINT USING, EOF, LOC, KILL, NAME.

INPUT. Cette instruction,dont la syntaxe est :

INPUT#N, VARIABLE

lit une donnée du fichier numéro N et l'affecte à la variable spécifiée. Par exemple, à l'exécution de la ligne INPUT #1,B$, le contenu de l'enregistrement « courant » (dans les fichiers séquentiels, la lecture et l'écriture débutent toujours à l'enregistrement 1 et continuent en séquence) du fichier 1 est lu et affecté à B$. Dans cette instruction, le symbole # ne peut pas être omis.

LINE INPUT. La syntaxe complète de l'instruction est :

LINE INPUT#N,VARIABLE

Elle lit une ligne entière de caractères (jusqu'à 254). Cette instruction trouve une utilisation particulière lors de la lecture d'un programme (mémorisé sur disque en format ASCII) par un autre programme. Le programme qui s'exécute voit le premier (celui sur disque) comme un fichier quelconque de données et peut donc le lire. Le seul obstacle réside dans la longueur variable des lignes du programme mémorisé en ASCII : chacune tient un nombre de caractères dépendant de l'instruction écrite ; on reconnaît le point où la ligne prend fin en lisant séquentiellement les enregistrements (caractères) jusqu'à rencontrer le code CR (Carriage Return) qui termine la ligne Basic.

WRITE. La fonction WRITE, dont la syntaxe est

WRITE#N,VARIABLE

écrit, dans le fichier identifié par le numéro N, la valeur contenue dans la variable spécifiée. Pour utiliser l'instruction, le fichier doit être ouvert en mode « O » (il s'agit d'une sortie du programme). L'instruction WRITE insère automatiquement une virgule entre les valeurs s'il y a plus d'une variable à transférer.

PRINT ET PRINT USING. La syntaxe de ces instructions est la suivante :

PRINT#N, VARIABLE
PRINT#N,USING...

Elles ont la même signification que les fonctions d'impression du même nom. Le fichier N est ici traité comme un périphérique.

EOF(N). Il s'agit d'une fonction de contrôle du fichier séquentiel numéro N, qui fournit la valeur – 1 (vrai) quand, lors de la lecture, on rencontre le code particulier signalant la fin du fichier. La fonction EOF sera utilisée en particulier dans la boucle de lecture d'un fichier séquentiel quelconque : elle permet le déroulement de la boucle, en lisant tous les enregistrements écrits, sans qu'il soit nécessaire de connaître au préalable la longueur du fichier. Par exemple, le programme

```
10 OPEN"I",1,"A:ESSAI"
20 INPUT#1,A$,B$,C$
30 IF EOF(1)GOTO100
40 GOTO 20
100 PRINT"FIN DE LECTURE DU FICHIER"
110 END
```

exécute la boucle constituée des instructions 20, 30 et 40 jusqu'à la rencontre du code End Of File. La fonction EOF est alors vraie et l'exécution se poursuit avec l'instruction 100. Si la ligne 30 était omise, la boucle ne serait plus contrôlée et tenterait de lire des renseignements au-delà des limites du fichier, provoquant l'apparition d'erreurs système et l'arrêt de l'exécution.

LOC(N). Fournit le nombre de secteurs lus ou écrits dans le fichier N à partir de son ouverture (rappelons qu'un secteur est constitué de 128 ou 256 octets).

KILL. L'instruction KILL"NOM" efface le fichier NOM, qui doit avoir été préalablement fermé. Elle est la même pour tous les types de fichiers (séquentiel, en accès direct au programme). Le fichier n'est pas physiquement « effacé » mais l'espace qu'il occupait est rendu disponible, et il pourra être écrasé par un nouveau fichier.

NAME. Cette fonction est utilisée pour renommer un fichier. L'instruction

NAME"NOM1" AS "NOM2"

remplace l'ancien nom (NOM 1) par le nouveau (NOM 2). Par exemple, la ligne NAME"OLDFIL" AS "NEUF" renomme le fichier OLDFIL sous le nom NEUF.

La page 640 illustre l'organigramme d'un programme qui utilise les instructions mentionnées afin d'écrire une série de données saisies au clavier sur un fichier séquentiel. Au terme de la saisie, le programme donne la longueur du fichier (en secteurs), puis relit et imprime les données saisies.

L'organigramme, qui peut paraître succinct par rapport à ceux présentés jusqu'ici, reflète la forme couramment utilisée dans la pratique. Pour être lisibles, les organigrammes d'un programme doivent synthétiser les procédures à accomplir, même sans détailler les moyens pour arriver au résultat : ce niveau de détail sera atteint dans la phase d'écriture des instructions proprement dites.

Par exemple, le pavé 500 de l'organigramme page 640 teste la fin de la phase de saisie. Les modalités de ce contrôle ne sont pas spécifiées de manière à s'appliquer à n'importe quel fichier. Dans le cas présent, chaque donnée se compose de quatre valeurs numériques et d'une chaîne de caractères ; le test portera sur cette dernière.

De même, pour relire le fichier qui vient d'être créé (et est donc ouvert en mode écriture), il faut tout d'abord le refermer, puis l'ouvrir à nouveau en mode lecture.

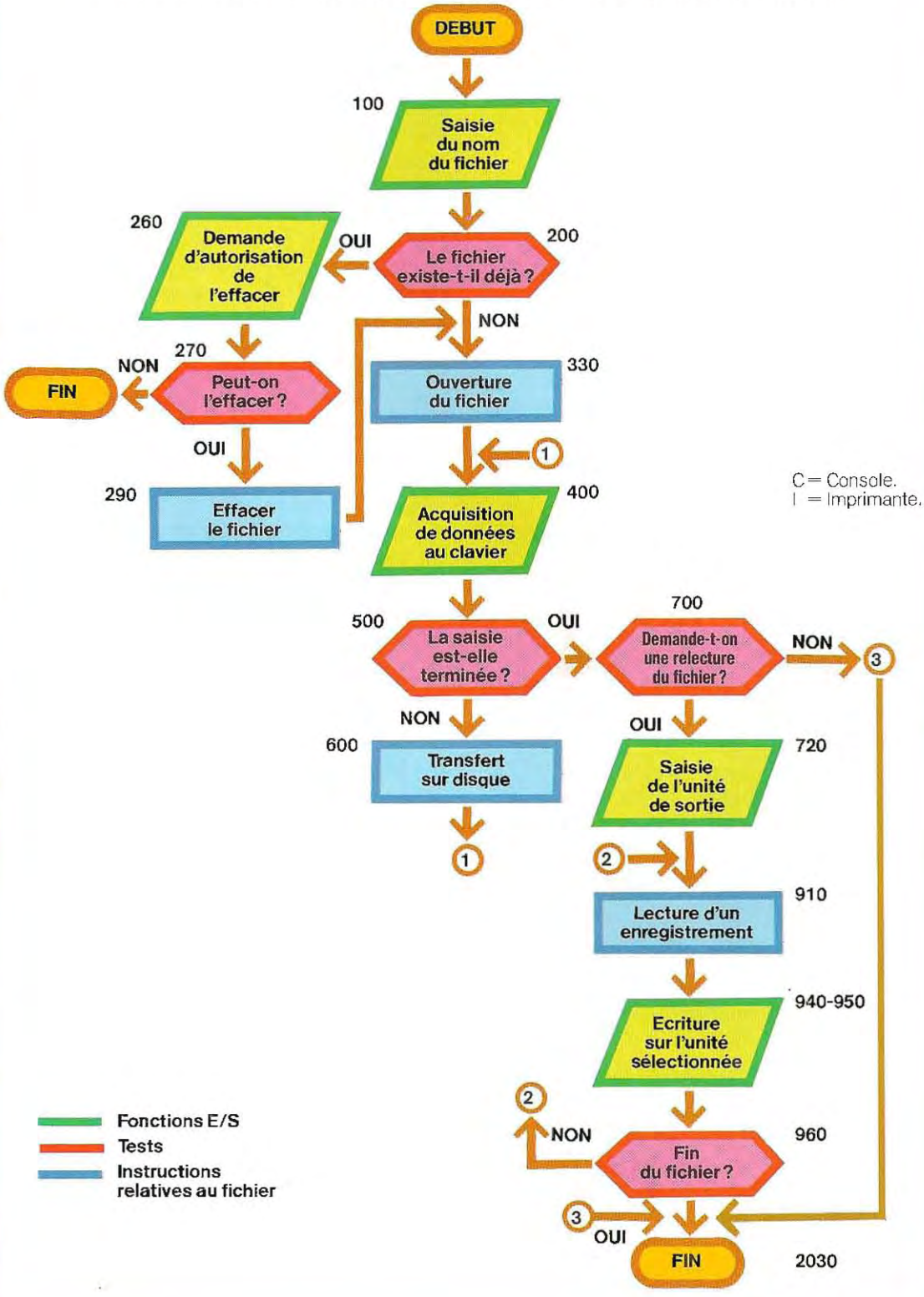
Pour contrôler si le fichier existe déjà, il suffit de l'ouvrir en lecture ; s'il n'est pas présent sur le disque, le système détecte une erreur, que le programme se charge de gérer.

Le programme est listé à la page 641. L'organigramme peu détaillé de la page 640 laisse une grande liberté de choix quant au moyen de codifier le programme. Le listing présenté propose une des solutions possibles, commentée et simplifiée intentionnellement. On peut toutefois obtenir les mêmes résultats avec des programmes plus compacts, mais moins compréhensibles.

Insertion de données dans un fichier séquentiel

Les enregistrements d'un fichier séquentiel ne sont pas directement adressables, on ne

EXEMPLE DE LECTURE ET D'ECRITURE D'UN FICHIER SEQUENTIEL



EXEMPLE DE LECTURE ET D'ECRITURE D'UN FICHIER SEQUENTIEL

```
10 ' *EXEMPLE DE LECTURE ET D'ECRITURE D'UN FICHIER SEQUENTIEL *
20 '   PROGRAMME SEQ
30 '
40 ON ERROR GOTO 1000
50 '
100 INPUT "NOM DE FICHIER";NM$
110 '
200 ' *Verification de son existence *
210 OPEN "I",1,NM$ 'On associe le numero 1 au fichier
220 '  Si le fichier n'existe pas, le systeme diagnostique une erreur
230 '  et l'execution se poursuit en 1000 (cf. ligne 40)
240 '
250 PRINT "LE FICHIER EXISTE DEJA SUR DISQUE"
260 INPUT "DOIT-ON L'EFFACER (OUI/NON)"; REP$
270 IF REP$<>"OUI" GOTO 2020
280 CLOSE 1
290 KILL NM$          'On ferme et on efface le fichier
300 OPEN "O",1,NM$ 'On ouvre le nouveau fichier en ecriture
310 PRINT
320 PRINT "LE FICHIER EST OUVERT EN ECRITURE"
330 PRINT "Pour terminer la saisie, taper FIN"
340 PRINT
400 INPUT "DONNEE (1 chaine et 4 nombres)",A$,N1,N2,N3,N4
500 IF A$="FIN" GOTO 700
600 WRITE #1,N1,N2,N3,N4,A$ 'Ici, le symbole # equivaut à #
610 GOTO 400
620
630 '*fin de la saisie *
700 INPUT "VOULEZ-VOUS UNE RELECTURE DU FICHIER (OUI/NON)";REP$
710 IF REP$="NON" GOTO 2020
720 PRINT "Ecriture sur quel peripherique (O = console, I = imprimante)"
730 INPUT REP$
740 IF REP$ <> "O" AND REP$ <> "I" THEN PRINT "ERREUR": GOTO 720
800 CLOSE 1                    'Fermeture du fichier en ecriture
810 OPEN "I",1,NM$             'Reouverture en lecture
820 K=0                        'Initialisation du compteur d'enregistrements
910 INPUT #1,N1,N2,N3,N4,A$ 'Lecture d'un enregistrement
920 K=K+1                      'Incrementation du compteur
930 PRINT "ENREGISTREMENT No ";K
940 IF REP$="O" THEN PRINT N1,N2,N3,N4,A$
950 IF REP$="I" THEN LPRINT "ENREGISTREMENT No";K: LPRINT N1,N2,N3,N4,A$
960 IF EOF(1) THEN 2000 'Fin du fichier
970 GOTO 910
980 '
1000 ' *Gestion des erreurs
1010 PRINT " LE FICHIER N'EXISTE PAS SUR DISQUE, IL DOIT ETRE CREE "
1020 RESUME 300
1030 '
2000 ' *SORTIE DU PROGRAMME *
2010 PRINT " LE FICHIER CONTIENT ";K;" ENREGISTREMENTS "
2020 PRINT " ***** FIN D'EXECUTION ***** "
2030 END
```

```
ENREGISTREMENT No 1
 12            45            78            23            MICHEL
ENREGISTREMENT No 2
 55            79            5             789           CHARLOTTE
ENREGISTREMENT No 3
 11            22            33            44            GEORGES
ENREGISTREMENT No 4
 99            88            77            66            YANICK
LE FICHIER CONTIENT 4 ENREGISTREMENTS
***** FIN D'EXECUTION *****
```

peut donc pas ajouter de données en les écrivant simplement à la fin du fichier. Il faut alors effectuer une copie du fichier à modifier, en ajoutant les nouveaux enregistrements en phase d'écriture de la copie. Plus précisément, on procède comme suit :

1 / Ouverture du fichier d'origine en mode « I » ; (il sera lu par le programme.
2 / Ouverture d'un second fichier en mode « O » ; il contiendra les enregistrements recopiés sur le premier, plus les nouvelles données.
3 / Transfert du fichier d'origine vers le second ; on crée ainsi une copie des données dans le nouveau fichier, qui est en phase d'écriture.
4 / Ajout des nouveaux enregistrements au second fichier.
5 / Le premier fichier, n'étant plus nécessaire, est d'abord fermé puis effacé.
6 / On attribue le nom du premier fichier au second ; on obtient ainsi un fichier ayant le même nom qu'à l'origine, mais de longueur supérieure.

Fonctions d'accès aux fichiers en accès direct

Les fichiers en accès direct sont les plus utilisés, car ils permettent d'adresser librement l'enregistrement auquel on s'intéresse. De plus, ils sont mémorisés en binaire, et occupent donc, sur le disque, moins d'espace que les fichiers séquentiels, mémorisés en ASCII.
Lors de leur ouverture, on ne déclarera pas le type d'accès, car ils peuvent être adressés indifféremment en lecture ou en écriture.
Pour l'échange de données avec ces fichiers, le système d'exploitation utilise une mémoire tampon. Avant d'exécuter l'écriture, les valeurs à transférer doivent transiter d'abord par cette mémoire tampon, qui sera ensuite déchargée sur disque.
Inversement, les données à lire sont extraites du disque et stockées dans le buffer, puis transmises au programme qui les demande par des instructions appropriées. Il est important de remarquer que cette mémoire tampon, qui constitue une variable particulière dont le nom est défini dans le programme d'application, ne peut être utilisée que pour échanger des données avec le disque. Avant toute autre utilisation, les données qui y sont contenues doivent être transférées dans un autre espace mémoire identifié par un nom différent.
Les fonctions et instructions utilisées dans la gestion des fichiers en accès direct dont les suivantes :

OPEN, FIELD, CLOSE, PUT, GET, LOC, KILL, NAME.

Analysons dans le détail la syntaxe et l'effet de chaque instruction. Pour les instructions KILL et NAME, on se référera à ce qui a été dit au sujet des fichiers séquentiels.

OPEN. En voici la syntaxe :

OPEN"R",#N,"NOM",L

où
R indique qu'il s'agit d'un fichier en accès direct ;
N représente le numéro d'unité logique attribué au fichier par le programme (le symbole # n'est pas toujours nécessaire) ;
NOM est le nom du fichier, et peut indiquer aussi l'unité disque sur laquelle il se trouve (par exemple A:ESSAI) ; quand l'unité disque n'est pas spécifiée, le système prend par défaut la dernière unité disque (driver) sélectionnée.
L indique la longueur en octets des enregistrements logiques constituant le fichier.

Par exemple, l'instruction :

OPEN"R",3,"B:TEST,64

ouvre un fichier en accès direct appelé TEST, sur l'unité-disque B. Le numéro logique 3 est associé au fichier, qui est structuré en enregistrements logiques de 64 octets.
L'instruction OPEN peut aussi servir à créer des fichiers.
Ce système, avantageux par certains côtés, peut poser quelques problèmes. Les programmes les plus complexes manipulent souvent des fichiers stockés sur plusieurs disques différents. Si l'utilisateur oublie de remplacer le disque, l'instruction OPEN, qui sur le bon disque aurait seulement ouvert le fichier, en créera alors un nouveau, avec le même nom mais sur le mauvais disque.
Le programme de la page 641 illustrait une

méthode pour éliminer tout risque d'erreur : on ouvre tout d'abord le fichier comme un fichier séquentiel, en mode INPUT. S'il n'existe pas, cette instruction provoque une erreur système et arrête l'exécution du programme. Si le fichier existe, le système ne signale aucune erreur. Il faut alors fermer le fichier en annulant l'ouverture en mode séquentiel (uniquement faite pour cette vérification), et l'ouvrir à nouveau, en mode « R » (accès direct).
Voici un programme simple, illustrant la technique exposée :

```
10  ' Ouverture d'un fichier en accès direct
20  ON ERROR GOTO 1000
30  OPEN"I",1,"ESSAI"
40  CLOSE1
50  PRINT "Le fichier existe"
60  OPEN"R",1,"ESSAI",64
70  '
80  ' Programme d'application
90  '
100  END
1000  ' ERREUR
1010  PRINT "Le fichier n'existe pas"
1020  INPUT "Doit-il être créé ?";R$
1030  IF R$<>"OUI" THEN STOP
1040  RESUME 60
```

Dans les programmes d'application, les instructions relatives au fichier (lignes 1010, 1020, 1030 et 1040) seront conditionnées par le type d'erreur. A l'apparition d'une erreur système, la variable ERR contiendra le code relatif à l'erreur, et ERL le numéro de ligne qui l'a engendré (ERR et ERL sont des noms réservés du système).
Par exemple, en supposant que le système d'exploitation réponde ERR=53 (c'est le cas en CP/M), le programme précédent doit être modifié de la manière suivante :

```
1000  ' ERREUR
1010  IF ERR=53 AND ERL=30 GOTO 1030
1020  ON ERROR GOTO 0
1030  PRINT "Le fichier n'existe pas"
1035  INPUT "Doit-il être créé ?";R$
1040  ' IF R$<>"OUI" THEN STOP
1050  ' RESUME 60
```

La ligne 1010 contrôle le type d'erreur et son origine ; si la cause de l'erreur n'est pas l'absence du fichier, le contrôle passe à la ligne 1020, qui réactive le mécanisme de gestion des erreurs dans le système ; le diagnostic habituel est émis et le programme s'arrête.

FIELD. Attribue à un fichier (préalablement ouvert) la mémoire tampon à utiliser dans les opérations de lecture et d'écriture. La syntaxe est la suivante :

```
FIELD 1, 64 AS B$
```

Cette instruction réserve la variable B$ au fichier numéro N, qui se compose d'enregistrements de 64 octets.
La mémoire tampon ainsi définie sera le « véhicule » d'échange de données avec le disque ; B$ ne peut plus servir à d'autres fins et, en particulier, ne peut pas être utilisée par des instructions d'INPUT et d'affectations classiques. Le transfert successif des données depuis le buffer vers le programme s'effectuera par les instructions LSET et RSET.
L'enregistrement logique peut être subdivisé entre plusieurs mémoires tampon.
Par exemple :

```
FIELD 1,12 AS N$, 30 AS C$, 20 AS V$, 2 AS F$
```

L'enregistrement (64 octets) sera réparti dans quatre buffers : N$ de 12 octets, C$ de 30 octets, V$ de 20 octets et F$ de 2 octets.

CLOSE. Fonctionne comme dans le cas des fichiers séquentiels : en exécutant l'instruction CLOSE#N, le fichier numéro N (précédemment défini par une instruction OPEN) est déclaré fermé et son contenu n'est plus accessible (jusqu'à une nouvelle ouverture du même fichier). L'instruction CLOSE libère les mémoires tampon intermédiaires attribuées au fichier ; elles pourront donc être utilisées de nouveau pour un autre fichier.

PUT. Instruction de transfert du contenu des mémoires tampon vers le disque. Il faut déclarer le fichier sur lequel on souhaite écrire, en spécifiant son numéro d'unité logique (défini par l'instruction OPEN) et l'emplacement (le numéro d'enregistrement logique). Par exemple, l'instruction

```
PUT#1,75
```

écrit sur l'enregistrement 75 du fichier 1 (dans certains cas le symbole # peut être omis).

Solutions du test 18

1 / Les attributs sont des indicateurs, stockés en mémoire en même temps qu'un caractère, et qui définissent certaines particularités de ce caractère. Pour cette raison, la mémoire associée à l'écran est plus étendue que celle qui serait strictement nécessaire à la définition en code ASCII des 24 × 80 caractères qui le composent.

2 / Les lignes 20 et 30 sont incorrectes. La ligne 20 affiche vingt fois une chaîne longue de 5 caractères (ESSAI), donc un total de 100 caractères sur la même ligne de l'écran. Ce dernier ne peut normalement contenir que 80 caractères par ligne. La ligne 30 contient le même type d'erreur : TAB(100) fait débuter l'affichage de la variable C en colonne 100, qui n'existe pas à l'écran (l'instruction serait valide s'il s'agissait de l'imprimante, LPRINT remplaçant alors PRINT).

3 / La boucle affiche une suite de données sur chacune des lignes de l'écran, jusqu'à la ligne 30. Puisque l'écran ne possède normalement que 24 lignes, les premières valeurs seront perdues puisqu'elles seront repoussées vers le haut (en dehors de l'écran) pour laisser place aux dernières lignes.

4 / Voici l'illustration du programme :

```
10 ' Solution question 4
20 ' Lecture des données au clavier
30 INPUT " Chaîne à écrire ";A$
40 INPUT " Numéro de colonne ";X
50 IF X=0 GOTO 160
60 INPUT " Numéro de ligne ";Y
70 IF Y=0 GOTO 160
80 '
90 ' Contrôles
100 '
110 K=LEN(A$) ' Longueur de la chaîne à écrire
120 IF (K+X)>80 THEN PRINT "ERREUR": GOTO 40
130 '
140 PRINT FNP$(X,Y);A$
150 GOTO 30
160 END
```

5 / L'instruction 140 devient :

```
140 PRINT   FNP$(X,Y);   INV$;   A$;   NOR$
            (Position)   (Vidéo inverse)   (Donnée)   (Retour affichage normal)
```

L'instruction pourrait être aussi écrite sur plusieurs lignes (le symbole ; supprime le retour en début de ligne). Elle pourrait, par exemple, être remplacée par :

```
140 PRINT FNP$(X,Y); 'Position
142 PRINT INV$;      'Affichage inversé
144 PRINT A$;        'Chaîne à écrire (donnée)
146 PRINT NOR$       ' Affichage normal
```

De cette façon, on peut conditionner certains ordres d'éditions, par la valeur d'un indicateur, afin d'obtenir des présentations différentes selon la valeur du flag. Par exemple, les instructions qui suivent génèrent l'édition en vidéo inverse dans le cas où FLAG=1 :

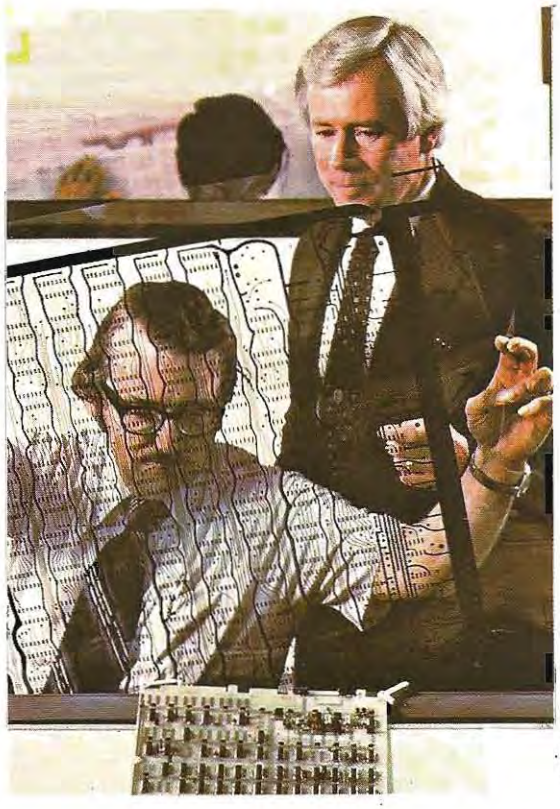
```
140 PRINT FNP$(X,Y);
142 IF FLAG=1 THEN PRINT INV$;
144 PRINT A$;
146 IF FLAG=1 THEN PRINT NOR$
148 IF FLAG<>1 THEN PRINT
```

La ligne 146 ramène en affichage normal et effectue le retour en début de ligne (pas de ;) dans le cas où FLAG=1. Dans le cas contraire (FLAG<>1), une ligne ne contenant que l'instruction PRINT est nécessaire pour obtenir le retour, qui n'avait pas été effectué par la ligne 144.

La taille d'un fichier en accès direct n'a pas besoin d'être définie a priori ; le système prend en compte les nouveaux enregistrements au fur et à mesure de leur écriture, et met à jour les pointeurs du fichier.
En général, ce procédé ne pose aucun problème, sauf si l'on modifie la taille des fichiers plusieurs fois. Au début, les fichiers sont ouverts et occupent physiquement des zones contiguës du disque. Si l'on désire rallonger le premier au-delà d'une certaine limite, les nouveaux enregistrements logiques ne pourront se placer de façon contiguë à la suite des précédents. Aussi, l'extension du premier fichier doit-elle se faire en un emplacement séparé. Les différentes parties de chaque fichier seront retrouvées par le système d'exploitation grâce à une table des extensions.
Entièrement gérée par le système d'exploitation, cette méthode s'avère parfaitement satisfaisante sur les micro-ordinateurs. En revanche, lorsqu'on s'adresse à de grosses machines, et pour des applications manipulant de très longs fichiers, il vaut mieux limiter cette dispersion des données.
On peut envisager deux solutions :

- copier périodiquement le disque contenant les fichiers sur un nouveau disque. A l'occasion du transfert, le système assure la contiguïté physique de tous les enregistrements d'un même fichier (extensions comprises) ;
- créer dès le début des fichiers de longueur égale à la taille maximum prévue ; ceci implique d'écrire d'une valeur de fin de fichier dans le dernier enregistrement prévu.

Projet de circuit imprimé.



Marka

Quelques kilo-octets pour jouer

Au début des années soixante-dix, alors que l'intérêt pour les OVNI tournait à la psychose et que l'on parlait tant des petits hommes verts, de leur présence présumée et de rencontres du troisième type, le premier « alien » véritable arriva sur terre.
Il s'appelait « Pong » et c'était un ***jeu vidéo*** *qui, grâce à un petit ordinateur, simulait pour la première fois une partie de ping-pong, en la représentant sur un moniteur de télévision.*
Ses apparences n'avaient rien d'anthropomorphique, mais il s'entendit immédiatement avec les habitants de notre planète, instaurant avec eux un rapport de paix et d'amitié. Il attira d'abord l'attention de cette faune humaine bariolée qui fréquente les salles de jeux, celle des « arcades » aux Etats-Unis, au point que les machines à sous furent en permanence saturées par un flot continu de pièces de monnaie : il connut un incroyable succès et sa progéniture, depuis l'Amérique, colonisa le globe entier.

Le premier à constater cet engouement fut précisément l'inventeur de Pong : Nolan Bushnell. Il ne perdit pas de temps et, poussé par un esprit d'entreprise authentique et génial, fonda Atari, première grande société de jeux vidéo de taille industrielle.
L'expansion extraordinaire des jeux électroniques conduisit les dirigeants d'Atari à lancer sur le marché une nouvelle version de Pong, destinée aux familles, qui utiliseraient justement leur téléviseur habituel comme moniteur.
On était en 1975, et Pong fut le premier jeu vidéo de l'Histoire.
Cette nouvelle présence quotidienne dans la maison bouleversa les rapports établis entre les téléspectateurs et la télévision : pour la première fois, on voyait à l'écran autre chose que la simple reproduction intangible de la réalité.

A partir de ce moment, le lien immatériel reliant l'émetteur à chaque poste de télévision était pour la première fois rompu, et une partie du monopole de l'information, qui créait un lien unidirectionnel entre la télévision et le spectateur, fut entamée.
Ce dernier, grâce au jeu vidéo, put ouvrir largement la fenêtre de l'écran sur une nouvelle réalité.
Le joueur interagit avec son partenaire électronique, échange avec lui des informations de façon continue et, à travers une représentation symbolique, ressent des émotions bien réelles : il devient acteur, propulsé sur la scène de l'écran.
Pour la première fois, s'établit ainsi un lien fonctionnel, unissant étroitement les deux biens de consommation les plus caractéristiques de notre époque : la ***télévision****, déjà détentrice d'une solide tradition auprès de millions de familles, et le* ***micro-ordinateur****. A l'époque, ce dernier effectuait ses premiers pas sur un marché encore vierge, mais prometteur, en particulier dans le domaine des jeux vidéo.*
Ce terme (néologisme dérivé de l'expression anglaise « video-game ») désigne un système électronique apte à reproduire à l'écran l'atmosphère et les circonstances d'un jeu donné. A partir de certaines règles de décision et des stimuli externes, (instructions du joueur), l'ordinateur simule, en temps réel, une modification de l'environnement du jeu, et la traduit à l'écran. Dans le jeu vidéo, on retrouve les caractéristiques fonctionnelles typiques d'un logiciel interactif.

Habituellement, un jeu vidéo se compose d'une console, de quelques dispositifs de commande, et d'un poste de télévision normal pour l'affichage. On sait qu'entre l'ordinateur et le joueur s'établit un dialogue, sous la forme d'un échange continuel d'informations. Les caractéristiques fonctionnelles des différents composants d'un jeu vidéo découlent logiquement de cette ***interactivité****.*
Une fois effectuée sa connection, l'ordinateur envoie au joueur la première information. Un peu comme pour les dessins animés, il génère un paysage de fond, sur lequel se déplacent les ***opérateurs actifs*** *(personnages) du jeu ; l'image ainsi synthétisée est envoyée sur l'écran de télévision.*
Réagissant à cette stimulation, le joueur entre en relation avec l'ordinateur, en accord avec la dynamique du jeu, et lui transmet ses décisions à travers les mécanismes de contrôle (joystik, trackball), exactement comme un conducteur tourne le volant, écrase la pédale d'accélérateur ou de frein. Les commandes,

saisies sous forme de déplacements mécaniques, sont converties par une interface en impulsions électriques, facilement interprétées par l'ordinateur en quelques millisecondes. Celui-ci vérifie leur compatibilité avec le déroulement correct du jeu et, se fondant sur les instructions de son programme de gestion, propose au joueur une image vidéo représentant une nouvelle situation de jeu. Dans une succession continuelle d'actions et de réactions, le logiciel déplace les opérateurs actifs, décrit de nouvelles scènes et de nouveaux paysages, modifie les couleurs, émet des sons ou de la musique, afin de rendre le jeu plus attractif.
Selon le type d'image proposée, on distingue deux catégories de jeux.
En reprenant l'exemple du conducteur, on appelle **subjectif** le jeu où le mécanisme de contrôle ne déplace pas des opérateurs actifs, mais tout le panorama de l'image télévisée, exactement comme si le joueur se trouvait à bord d'une voiture ou d'un avion (la célèbre simulation de vol) ; dans le jeu **objectif**, l'opérateur actif représente à l'écran le joueur, celui-ci s'identifiant en le faisant bouger selon ses commandes.

Tout ordinateur de jeu vidéo est capable de gérer les deux catégories, moyennant le simple remplacement de la **cartouche** (cassette) où est mémorisé le logiciel approprié.
La cartouche a constitué, pour le marché du jeu vidéo, un élément révolutionnaire, clé de son succès commercial. Grâce à elle, l'utilisateur n'est plus contraint à changer de console pour changer de jeu : dans le cas de Pong, toutes les données du programme résidaient dans une mémoire indissociable de la console. Dans les jeux vidéo des générations suivantes (le VCS 2600 d'Atari, par exemple), le logiciel réside sur une mémoire dissociée de la console : un connecteur permet de la remplacer par d'autres cartouches, et d'accéder ainsi à de nouveaux jeux.
Cette interchangeabilité, comparable à celle d'un magnétophone à cassettes, a contribué à développer un nouveau marché, celui de la production de cartouches pour jeux vidéo. Accessible aux sociétés qui ne fabriquaient pas de consoles, ce marché a assuré la naissance et la prospérité de nouvelles industries, qui aujourd'hui peuvent se vanter de chiffres d'affaires annuels de centaines de millions de dollars. La profession de « game designer » (créateur de jeux vidéo) s'installe dans la notoriété.
Comme ses collègues artistes et musiciens, celui-ci signe ses propres créations, participe aux bénéfices des ventes, possède son public d'admirateurs et devient la vedette des années quatre-vingts.
Pour lui, concevoir un jeu vidéo à partir d'un sujet original constitue un métier très lucratif et, pour la société qui produit la cartouche, une affaire de millions de dollars : un nouvel Eldorado !

Mais les routes qui conduisent à cette mine d'or commencent en 1983 à être encombrées. L'envie de s'approprier une petite tranche des 75 milliards de francs qui représentent le chiffre d'affaires de l'industrie de jeux vidéo a poussé une multitude de petites et grandes sociétés à se lancer dans l'aventure.
Ce fut inévitablement le conflit. Piégées dans une course frénétique pour la conquête d'un marché, jusque-là considéré comme facile et prometteur, mais qui s'avéra bientôt plus complexe que prévu, de nombreuses sociétés se sont retrouvées au bord de la catastrophe.
Au début de 1983, l'avènement de la troisième génération de jeux vidéo a provoqué une reconfiguration draconienne des parts du marché, rendant difficile la survie de ceux qui ne pouvaient offrir un produit technologiquement à jour.
Parallèlement, s'est ouvert une polémique violente au sujet du danger présumé des jeux vidéo sur le plan psychologique. Impitoyablement, certains mass média menèrent une campagne de scandales, afin de discréditer le fétiche moderne qui, quelques mois auparavant, était l'objet d'un tel engouement. Ainsi s'éteignit un phénomène caractéristique de cette culture des excès, où les choix dédaignent parfois l'intelligence et la capacité de réflexion reconnues essentiellement humaines, et dérivent d'un sentiment d'angoisse collective, à laquelle la société croit échapper par une consommation effrénée d'artifices fascinants, colorés, séduisants, mais terriblement éphémères.
Au cours de l'année 1983, à la Bourse de New York, on a respiré, pendant plusieurs jours, un

air de tempête : certaines grosses sociétés chancelaient sous les coups infligés par les pertes de centaines de millions de dollars, d'autres perdaient pied, suivies du glas funèbre de la cloche des Lloyds. En 1983, le jeu vidéo nous a montré ce sinistre paysage, et de nombreux personnages furent désintégrés pour disparaître de la scène ; cependant, la partie ne s'est pas arrêtée sur un catégorique « game-over ». Les ventes de consoles et de cartouches ont subi un certain fléchissement, signe d'un plus grand discernement dans les achats ; mais le public n'a pas déserté massivement.

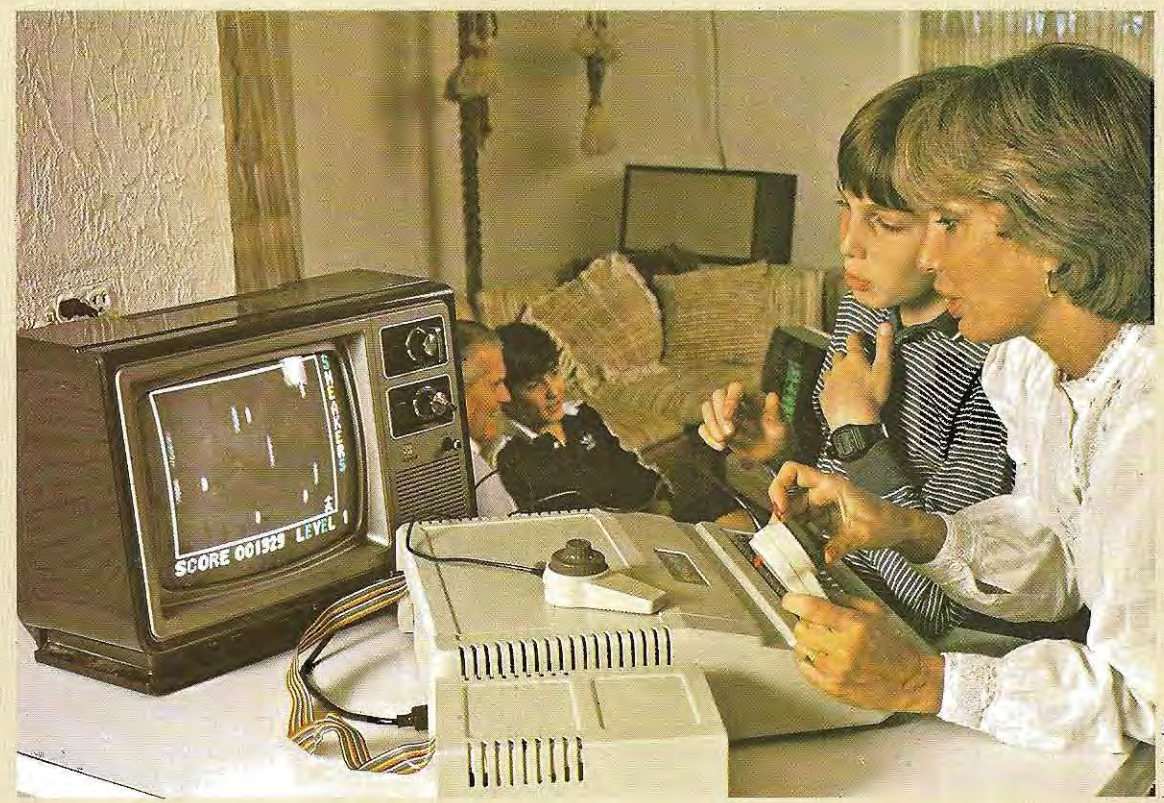
Aux Etats-Unis, la recette des arcades a dépassé les 5 milliards de dollars, franchissant de manière significative le chiffre d'affaires des industries du cinéma et du disque. Nous sommes en présence d'un phénomène ancré désormais dans le quotidien des habitants d'une bonne partie du monde, mais enfin reconnu pour ce qu'il est : une source d'émotions à bon marché.

Les aventures offertes au joueur varient à l'infini et satisfont tous les goûts. Les statistiques affirment qu'aux Etats-Unis le rapport entre la vente de cartouches et de consoles s'établit, pour l'utilisateur moyen, aux alentours de sept pour un : sept jeux vidéo différents par joueur. On lui propose d'évoluer entre des guerres stellaires (application typique de l'idée : faire feu et s'enfuir), d'échapper à des animaux et fantômes féroces, de trouver sa route dans des labyrinthes inextricables, de sauver la terre d'envahisseurs inter-galactiques. Le joueur peut s'asseoir à la cabine de pilotage d'un B 17, et, depuis son fauteuil, envoyer tranquillement des bombes sur l'Allemagne nazie ; en fait, pas vraiment tranquillement, car un synthétiseur vocal ou un générateur de sons reproduit au mieux les sensations et les peurs qu'un pilote est susceptible d'éprouver dans un bombardier en mission. Des hurlements stridents, des appels au secours, des instructions et des ordres donnés sèchement se succèdent à un rythme serré. Le pilote joueur doit prendre des décisions immédiates, au risque d'être abattu et de tomber dans les eaux de la Manche. Il est à la fois amusant et inquiétant de voir de paisibles fonctionnaires dans le rôle d'as de la RAF.

La recherche et l'expérimentation de nouveaux jeux vidéo se développent en mêlant des connaissances psycho - physico-sensorielles toujours croissantes. Les produits commercialisés aujourd'hui impliquent une technologie sophistiquée de synthèse et de traitement de l'image vidéo, alliant la micro-électronique à la technique du laser. Au moyen de dispositifs électroniques recourant largement aux composants modernes (microprocesseurs, etc.), l'**image analogique** est numérisée et rendue accessible à l'ordinateur, qui se charge de la transcrire par un rayon laser sur un disque spécial de métal et de plastique. Sur une face du disque, environ 250 000 images synthétiques distinctes peuvent être mémorisées. A la différence des bandes magnétiques, qui ne fournissent les images que dans un ordre strictement séquentiel, il est possible d'aller « lire » sur le disque, très rapidement, deux images différentes, stockées à des adresses très éloignées. Reprenons l'exemple du bombardier.

Les fabricants de jeux vidéo organisent régulièrement des expositions : ici, stand Mattel au Computer Play 83.

R. Lonardi/AICA



Imprimatur

Jouer avec un ordinateur signifie aussi apprendre.

Comme les images produites par les jeux vidéo sont très schématiques, on peut envisager, pour rendre la simulation plus vraisemblable, de projeter sur l'écran des séquences tirées d'un film de guerre, montrant une ou plusieurs scènes de bombardement. En manipulant les commandes du jeu vidéo, on aura alors l'impression d'être à bord d'un vrai B 17, mais qu'on ne pourra pas guider, puisqu'aucun rapport de cause à effet (ou mieux de commande à mouvement) ne peut s'établir entre les commandes et le film séquentiel : quoi qu'on fasse, le film se déroule toujours à son propre rythme.

Avec l'emploi du disque vidéo, au contraire, le logiciel choisit parmi toutes les quatre images, celle correspondant aux manœuvres et aux décisions du joueur pilote.
On crée de cette façon des parcours très réalistes fondés sur un véritable rapport commande-mouvement.
En manœuvrant les commandes, on verra défiler ***en temps réel*** *une suite d'images télévisées correspondant au style de pilotage : en tournant à gauche, on verra les maisons, la campagne, l'horizon (avec une précision cinématographique) se déplacer vers la droite.*
On pourra tirer sur les ennemis, viser en suivant la trace des projectiles et les voir exploser véritablement, comme si on était réellement aux commandes d'un avion de guerre ou dans le rôle d'un héros de film.
Le vieux rêve des instructeurs de vol de posséder un simulateur parfait est ainsi réalisé.

Ces systèmes sophistiqués ne demeurent cependant pas dans le domaine professionnel : le jeu vidéo du futur est maintenant à notre portée ; aux Etats-Unis, certains exemplaires de ce type de jeu sont déjà commercialisés.
« Dragon's Lair », le précurseur de cette nouvelle vague, a été réalisé avec les ***photogrammes*** *d'un dessin animé style Disney, et se distingue par un graphisme absolument remarquable.*
Il propose au joueur quarante deux scénarios différents, où se déroule une histoire digne de

la meilleure poésie épique médiévale : sauver la belle princesse prisonnière d'un dragon ; il y a deux cents décisions à prendre d'une manière correcte, sinon c'est la mort.

Moins chevaleresque mais plus dramatique, le scénario proposé à certains étudiants en médecine américaine remplace la belle princesse par un patient atteint de péritonite, et qui doit être opéré d'urgence.
Un contexte réel, qui nécessite la plus grande attention et des prises de décision rapides, apprend aux futurs médecins à opérer sans erreur, à travers une simulation d'une vraisemblance absolue.
La première application de ce nouveau jeu est donc ***didactique****, et gagnera certainement nos écrans dans un avenir proche.*

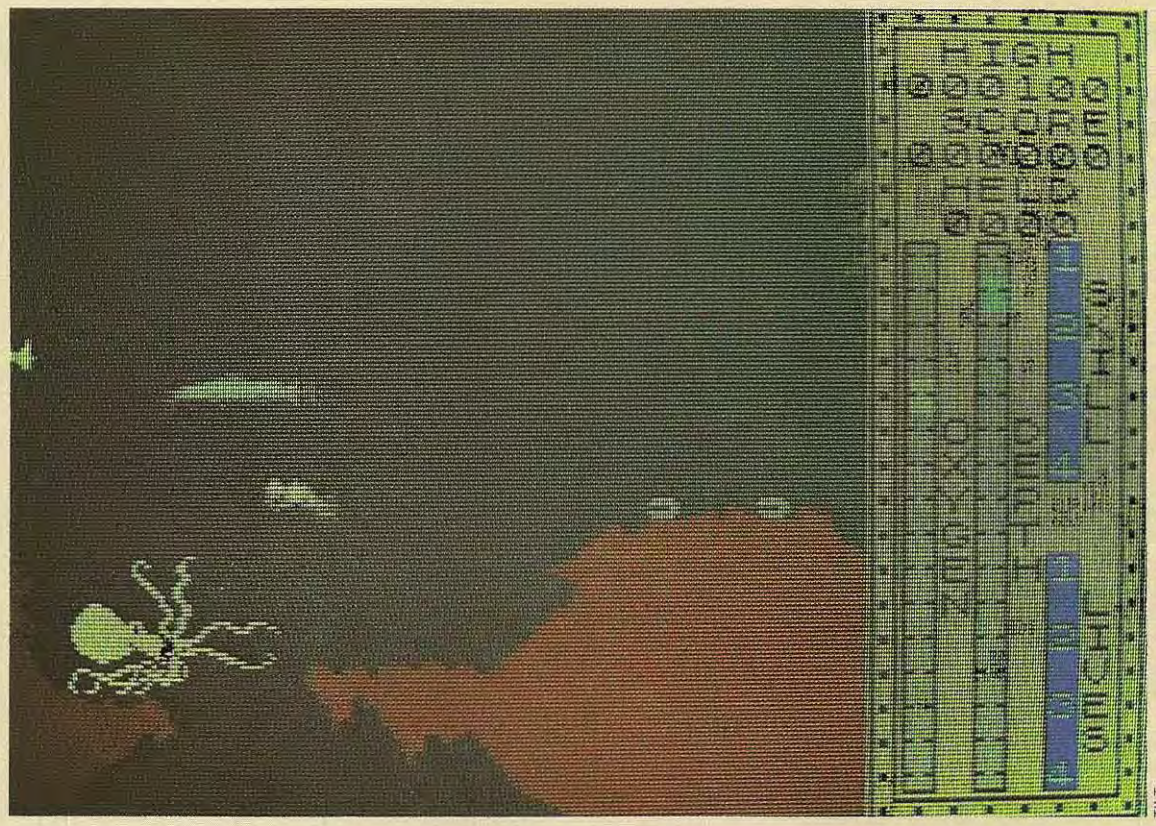
En 1984, des choix courageux s'imposent : on assiste à la naissance d'un phénomène qui, dans les années à venir, va marquer nos temps de loisirs.
Inutile de se leurrer quant à l'indépendance présumée du jeu vidéo par rapport aux modèles mentaux érigés par la société actuelle : il en constitue un fidèle reflet. Le milieu culturel engendre ses propres loisirs, et les jeux vidéo en témoignent.
Inversement, les schémas proposés par les jeux vidéo influent, dans une certaine mesure, sur le ***comportement*** *du joueur.*
Tant qu'on y verra presqu'exclusivement des messages d'agressivité et de guerre, nul ne s'étonnera de la violence latente qui caractérise notre société.
Remarquons toutefois qu'une alternative à l'agressivité engendrée par les jeux vidéo (guerres stellaires, etc.) est déjà proposée avec des jeux qui requièrent des qualités de rapidité de vision et d'action, et qui proposent au joueur de déjouer des séries de difficultés qui font de ces jeux des aventures passionnantes.

Pour certains, le jeu vidéo menace dangereusement l'activité intellectuelle : il est le produit de modèles mentaux utilisés d'une manière non critique, et proposés continuellement au joueur sous une forme obsédante et privée de sensibilité, car masquée par les aspects extérieurs les plus séduisants (couleurs, musiques, sons).

Il ne faut pas oublier que ces jeux s'adressent justement aux enfants et aux jeunes, dont la personnalité plus fragile risque de subir certaines distorsions dangereuses.
Dans les pays anglo-saxons, la vente des jouets obéit à une réglementation sévère : l'usage des jouets doit être précisé lisiblement sur l'emballage, accompagné d'une référence à l'âge des enfants ou des adolescents auxquels ils sont destinés.
Il s'agit là d'une forme très répandue de ***protection*** *et de tutelle vis à vis des plus faibles, au sein des pays les plus évolués socialement.*
Une formule analogue, concernant l'aspect pédagogique et le type de public auquel ils s'adressent, devrait être mise en place également dans un avenir proche pour les jeux vidéo.

Le jeu vidéo existe en tant que phénomène social et peut jouer un rôle culturel positif, surtout dans la mesure où il participe au développement de la propre ***créativité*** *du joueur.*
Jean Piaget affirmait qu'on apprend davantage lorsqu'on invente quelque chose ; en ce sens, le jeu vidéo et l'ordinateur domestique peuvent constituer deux instruments actifs de connaissance.
Les différences entre le jeu vidéo et l'ordinateur domestique ne sont pas si grandes ; ce dernier représente l'évolution logique du jeu vers des applications plus larges.
L'ordinateur possède la même structure de base interne qu'une console de jeu vidéo, améliorée par l'introduction d'autres dispositifs : un clavier, plus d'espace mémoire et un lecteur de cassettes pour stocker les programmes.
Certains constructeurs de jeux vidéo proposent déjà des « extensions informatiques » compatibles avec le système de base, accompagnées d'un manuel d'apprentissage du langage Basic.
Le jeu vidéo constitue ainsi une approche remarquable de l'informatique, surtout chez les enfants, en habituant l'utilisateur au dialogue avec l'ordinateur.

Depuis quelques années, on assiste dans toute l'Europe, à une véritable invasion d'ordinateurs domestiques.
Dans de nombreux pays, on offre un ordina-



Un exemple de jeu vidéo : ces aventures sous-marines sont mises en valeur par un excellent graphisme et une très bonne animation (Durell Software pour Spectrum 48 K).

teur comme cadeau de Noël à des jeunes dont l'âge se situe entre dix et quatorze ans. Parmi les programmes destinés aux ordinateurs domestiques, il est possible d'établir une classification des plus demandés : à travers celle-ci, on a mis aussi en évidence les critères d'application de l'ordinateur dans le domaine familial.

Les jeux de société et les jeux éducatifs prédominent toujours : ils couvrent 54 % des ventes. L'importance de ce phénomène n'est pas à sous-évaluer : le jeu est un choix qui persiste même quand le joueur peut créer ses programmes de manière autonome. Ceci signifie qu'on est en présence de ***stimulations*** *fondamentales, de désirs profonds qui s'expriment chez l'homme et qu'il veut satisfaire de toute façon, et pas seulement dans son plus jeune âge.*

Dans toute l'Europe, un certain nombre d'études ont été faites dans ce domaine, débouchant souvent sur des congrès et des expositions.

Ces démarches avaient en général pour but d'établir un débat sur les problèmes liés à la diffusion des jeux vidéo, et de proposer, de la part d'experts qualifiés, une analyse des nombreuses questions que l'avènement des jeux informatiques a soulevées : qu'est-ce qu'un jeu, comment le conçoit-on, comment l'évalue-t-on, quels sont les aspects éducatifs du jeu informatique, quel marché couvre-t-il, à travers quel jeu, par quelle méthode peut-on introduire le jeu dans l'enseignement, quel jeu proposer pour un âge donné.

Durant ces congrès, les participants ont affronté ces questions, en partant d'analyses différentes. On a, toutefois, pu mettre en évidence une tendance toujours plus prononcée vers une plus vaste application des jeux éducatifs à ***haut niveau conceptuel****.*

Cette nouvelle culture du jeu s'impose aux goûts des gens, et on doit lui prêter toute l'attention qu'elle mérite. Le but des promoteurs a été d'observer sur le terrain ce qui se fait, au niveau de l'initiative privée, dans le secteur des jeux pour ordinateur.

GET. La syntaxe de cette instruction est la suivante :

GET # N,R

Elle lit les données de l'enregistrement R du fichier N et les transfère dans les mémoires tampon précédemment associées au fichier N. Par exemple,

GET # 1,75

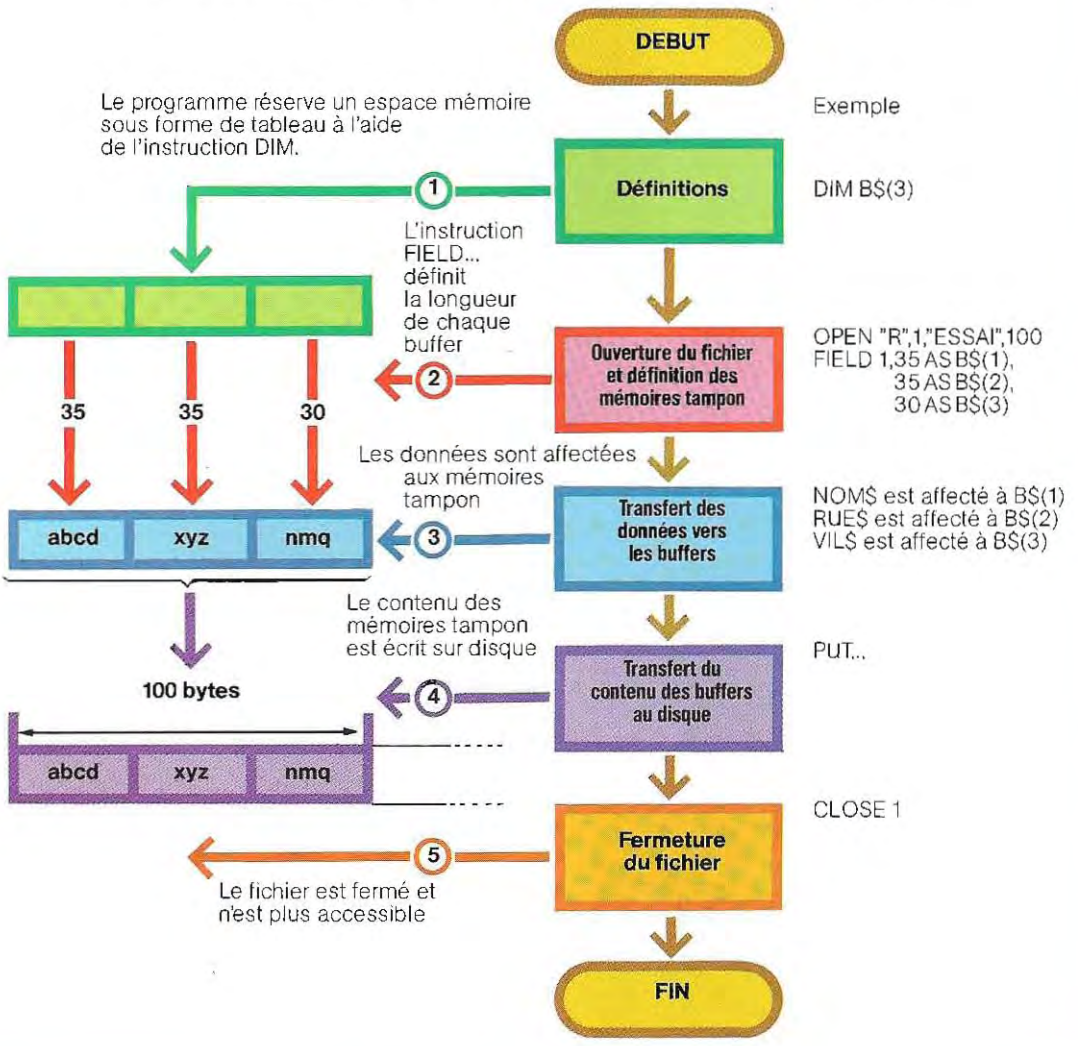
lit les données contenues dans l'enregistrement 75 du fichier 1.

LOC(N). La fonction LOC(N), où N représente le numéro d'unité logique associé au fichier, fournit la valeur « courante » du pointeur, c'est-à-dire le numéro du dernier enregistrement utilisé (dans une instruction PUT ou GET), augmenté de 1. Par exemple, l'exécution de la suite d'instructions :

```
10  GET 1,7
20  PUT 1,9
30  A=LOC(1)
```

affecte à A la valeur 10, puisque le dernier numéro d'enregistrement utilisé vaut 9 (instruction 20).
Le schéma ci-dessous illustre un transfert de

SCHEMA D'ECHANGE DE DONNEES AVEC LE DISQUE



PROGRAMME D'ECRITURE SUR UN FICHIER SEQUENTIEL

```
10  ' ** PROGRAMME D'ECRITURE SUR UN FICHIER SEQUENTIEL **
20  '
30  INPUT "NOM DU FICHIER ";NN$
40  NN$="A:" + NN$                  ' NN$ contient le nom complet du fichier
50  '                                (A est l'unite disque ou il est stocke)
60  '
70  OPEN "O",1,NN$                  ' Ouverture en mode ecriture du fichier de nom NN$
80  '                                et de numero logique 1
90  ' GOSUB 500                     ' Routine de saisie des donnees au clavier
100 CLOSE 1
110 '
120 OPEN "I",1,NN$                  ' Ouverture en mode lecture
130 GOSUB 1000                      ' Routine de relecture et impression
140 CLOSE 1
150 '
160 '
170 INPUT "Voulez-vous ajouter de nouvelles donnees (oui/non) ";REP$
180 IF REP$="NON" GOTO 400          ' Sortie du programme
190 '
200 OPEN "I",2,NN$                  ' Ouverture en lecture du fichier original NN$
210 '                                avec le numero logique 1
220 OPEN "O",1,"A:COPY"             ' Ouverture en ecriture d'un nouveau fichier
230 '                                avec le numero logique 1
240 INPUT £2,V,A$                   ' Lecture d'un enregistrement
250 PRINT £1,V,A$                   ' Copie sur le nouveau fichier
260 IF EOF(2) GOTO 280
270 GOTO 240                        ' Vers une nouvelle recopie
280 CLOSE 2                         ' Le fichier original (2) a ete entierement recopie
290 KILL NN$                        ' On l'efface
300 '
310 GOSUB 500                       ' Saisie des donnes a rajouter sur le fichier 1
320 CLOSE 1
330 NAME "A:COPY" AS NN$            ' On renomme le nouveau fichier avec
340                                  le nom original
350 OPEN "I",1,NN$
360 GOSUB 1000                      ' Relecture
370 CLOSE 1
380 GOTO 170                        ' Vers une nouvelle modification eventuelle
390 '
400 END
410 '
500 ' ** ROUTINE DE SAISIE DES DONNEES AU CLAVIER **
510 PRINT "POUR TERMINER LA SAISIE, TAPER 0"
520 PRINT
530 INPUT "Valeur numerique a ecrire ";V
540 IF V=0 GOTO 580          ' Sortie
550 INPUT "Chaine alphanumerique ";A$
560 PRINT £1,V,A$            ' Ecrit sur le fichier 1 les valeurs V et A$
570 GOTO 530                 ' Vers une nouvelle saisie
580 PRINT "**** FIN DE SAISIE ****"
590 RETURN
600 '
1000 ' ** ROUTINE DE RELECTURE ET IMPRESSION DU FICHIER **
1010 INPUT £1,V,A$           ' Lecture des donnees d'un enregistrement
1020 LPRINT                  ' et affectation aux variables V et A$
1030 LPRINT "Valeur numerique lue = ";V
1040 LPRINT "Chaine alphanumerique = ";A$
1050 IF EOF(1) GOTO 1070
1060 GOTO 1010               ' Vers une nouvelle lecture
1070 PRINT "**** FIN DE RELECTURE ****"
1080 RETURN
```

```
Valeur numerique lue = 1985
Chaine alphanumerique = ISABELLE

Valeur numerique lue = 1959
Chaine alphanumerique = LAURE

Valeur numerique lue = 1925
Chaine alphanumerique = ANNE
```

données sur disque pour la mémorisation d'un carnet d'adresses. Les champs prévus sont les suivants :

nom et prénom = 20 + 15 caractères ;
numéro et rue = 5 + 30 caractères ;
code postal et ville = 30 caractères.

On utilise 3 buffers, une pour chaque champ de données, qui peuvent être appelés par trois noms quelconques de chaîne. Il est cependant préférable d'employer toujours le même nom générique, différencié par un indice : on peut utiliser un tableau de chaînes, dimentionné dans ce cas particulier à trois valeurs, de façon à n'avoir qu'un seul nom réservé à la mémoire tampon, dans les opérations d'entrées/sorties.

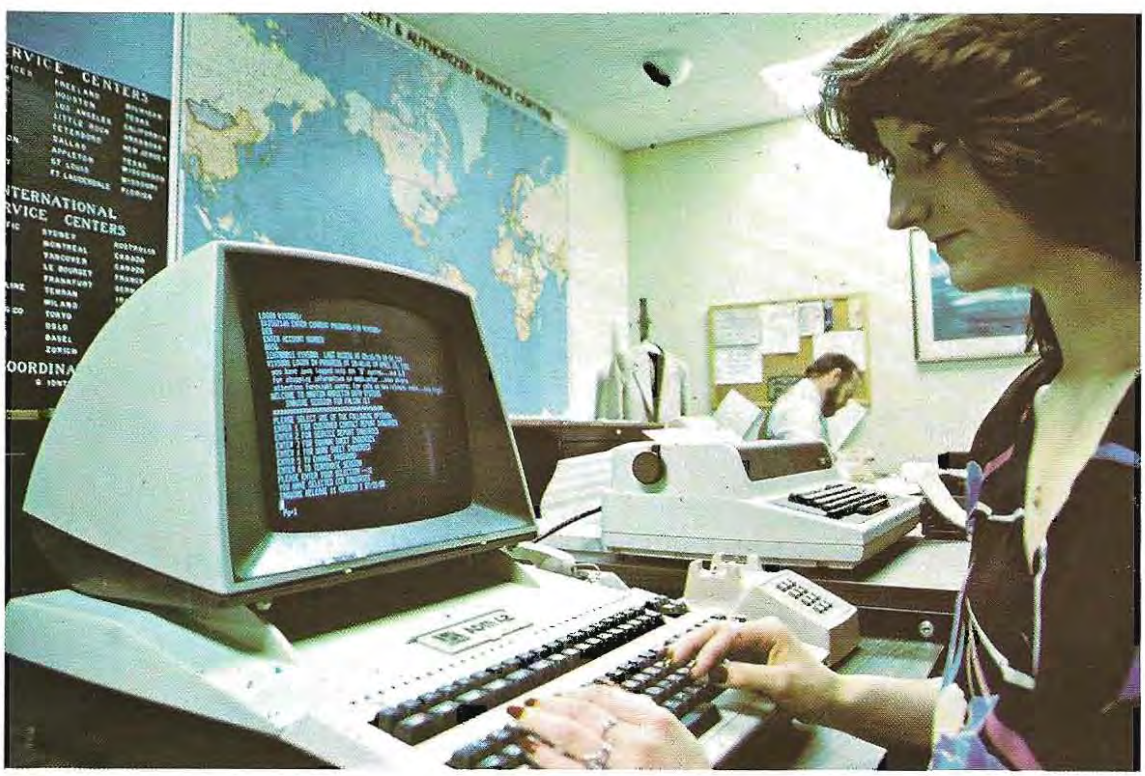
Formats et codifications des données

Dans les fichiers séquentiels, les données sont mémorisées en format ASCII, mais elles peuvent cependant être lues par le programme sous forme de valeurs numériques. Le programme listé à la page 653 écrit sur un fichier séquentiel (ouvert ligne 70 sous le numéro 1) un certain nombre de données saisies au clavier dans la routine 500, constituées alternativement d'une valeur numérique et d'une chaîne. La saisie se termine en entrant 0 comme valeur numérique (ligne 540). Pour relire le fichier et l'éditer sur l'imprimante (routine 1000), il faut d'abord le fermer puis l'ouvrir en mode « I ». On remarquera que les instructions PRINT et INPUT des lignes 560 et 1010 contiennent le symbole £1 (qui remplace ici le #1 classique), pour indiquer à l'interpréteur Basic qu'il s'agit d'un échange de données avec l'unité logique 1 et non avec la console.

Le programme propose ensuite de rajouter des données au fichier séquentiel ainsi créé. Dans les fichiers en accès direct, les données sont mémorisées sous forme binaire, mais elles doivent toujours être fournies en format ASCII, puisqu'on utilise comme support une mémoire tampon de type chaîne. Cela constitue la seule difficulté survenant dans les opérations d'E/S avec ce type de fichier ; le jeu d'instructions du Basic permet d'ailleurs de la surmonter facilement.

Les fonctions principales utilisées pour la

Un ordinateur ADM utilisé à la Falcon Jet Company.



Leo de Wys/Marka

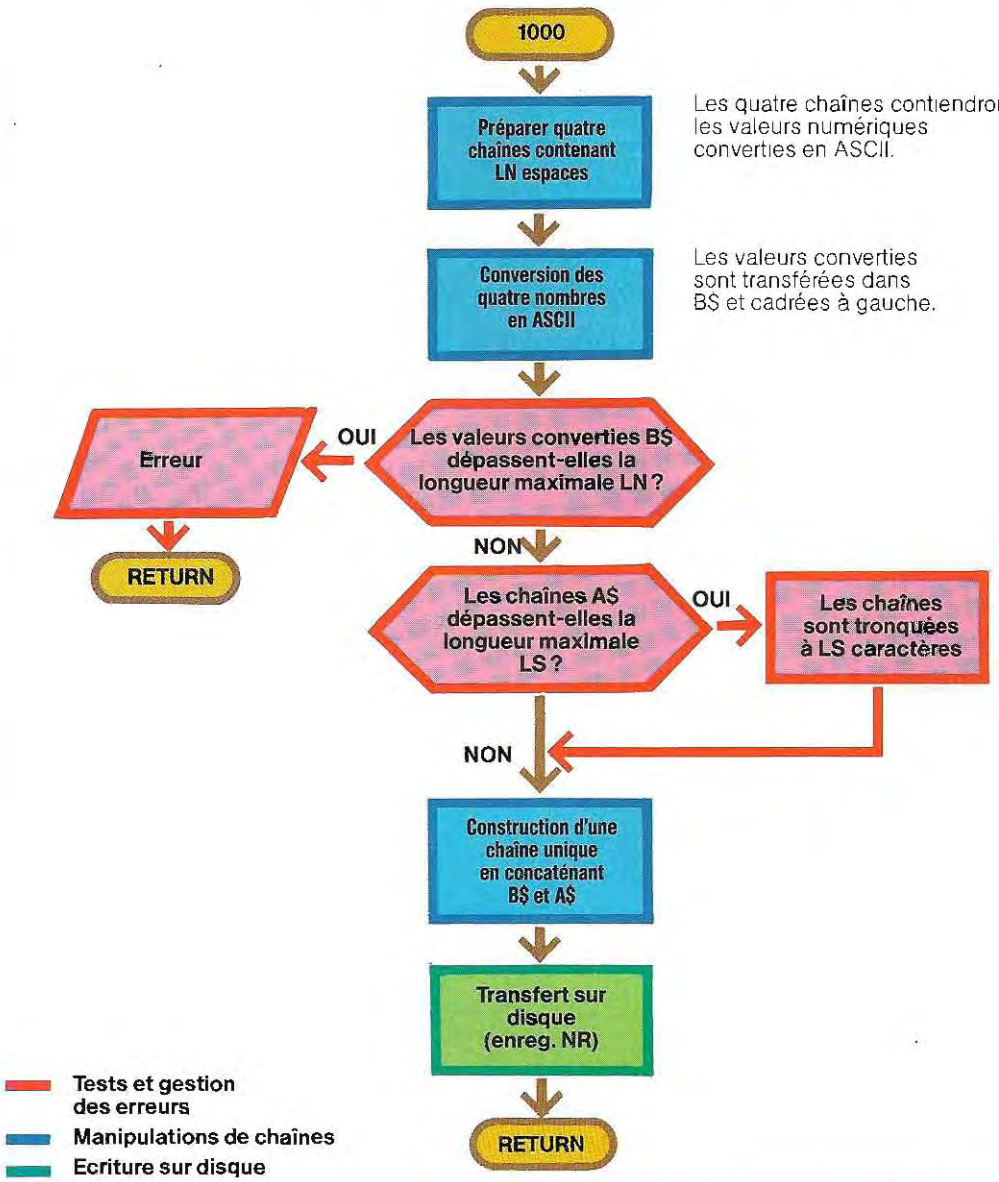
PROGRAMME D'ECRITURE SUR UN FICHIER EN ACCES DIRECT

```
5 ' ** PROGRAMME D'ECRITURE SUR UN FICHIER EN ACCES DIRECT **
10 ' Exemples de conversion des valeurs numeriques pour les operations d'E/S
20 ' avec le disque (fichiers en acces direct)
30 '
40 OPTION BASE 1
50 DEFINT I          ' Les noms d'initiale I se referent a des variables
60 '                   entieres
70 DEFSNG R          ' Les noms d'initiale R se referent a des variables
80 '                   en simple precision
90 DEFDBL D          ' Les noms d'initiale D se referent a des variables
100 '                  en double precision
110 DIM BF$(3)       ' Buffer d'entrees/sorties (un pour chaque type de variable)
120 OPEN "R",1,"TEST",30      ' Chaque enregistrement a une longueur de 30 octets
130 FIELD 1,10 AS BF$(1), 10 AS BF$(2), 10 AS BF$(3)
140 '
150 '
160 ' ** Lecture des valeurs numeriques a ecrire sur disque
170 '
180 INPUT "Valeur entiere"; INTE0
190 INPUT "Valeur en simple precision"; REEL
200 INPUT "Valeur en double precision"; DOUBLE
210 '
220 ' ** Les variables INTE0, REEL et DOUBLE contiennent les donnees saisies,
230 '    a convertir en ASCII
240 '
250 A1$=MKI$(INTE0)          ' Les fonctions MKI$, MKS$ et MKD$
260 A2$=MKS$(REEL)           ' transforment les valeurs numeriques
270 A3$=MKD$(DOUBLE)         ' en leur equivalent ASCII
280 '
290 ' ** Saisie du numero d'enregistrement a remplir
300 INPUT "Dans quel enregistrement doit-on ecrire"; IR
310 '                    le numero IR est un entier (initiale I)
320 '
330 ' ** Transfert dans les buffers d'E/S
340 '
350 BF$(1)=A1$           ' Rappelons que les variables intermediaires
360 BF$(2)=A2$           ' A1$, A2$ et A3$ sont necessaires, car aucune
370 BF$(3)=A3$           ' operation ne peut etre effectuee sur les buffers
380 '
390 ' ** Ecriture sur le fichier
400 ' Le contenu des buffers BF$ est ecrit dans l'enregistrement
405 ' numero IR du fichier 1
410 PUT #1,IR
420 INPUT "Voulez-vous continuer (oui/non)"; REP$
430 IF REP$="oui" GOTO 180        ' Nouvelle ecriture
440 '
450 ' ** Relecture du fichier
460 '
470 INPUT "Quel enregistrement voulez-vous relire"; IR
480 GET #1, IR          ' Lecture de l'enregistrement numero IR
485 '                     Les donnees sont transferees dans les buffers BF$
490 A1$=BF$(1)
500 A2$=BF$(2)
510 A3$=BF$(3)
520 INTE0=CVI(A1$)           ' Les fonctions CVI, CVS et CVD
530 REEL=CVS(A2$)            ' transforment les codes ASCII
540 DOUBLE=CVD(A3$)          ' en valeurs numeriques
550 LPRINT "Donnees lues : "; INTE0, REEL, DOUBLE
560 INPUT "Voulez-vous continuer (oui/non)"; REP$
570 IF REP$="oui" GOTO 470
580 CLOSE #1
590 END
```

transformation des données sont :
MKI$, MKS$, MKD$ en écriture (conversion de valeur numérique en chaîne) ;
CVI, CVS, CVD en lecture (conversion de chaîne en valeur numérique).
Le programme listé ci-dessus utilise ces fonctions pour l'écriture et la lecture de données sur un fichier en accès direct.

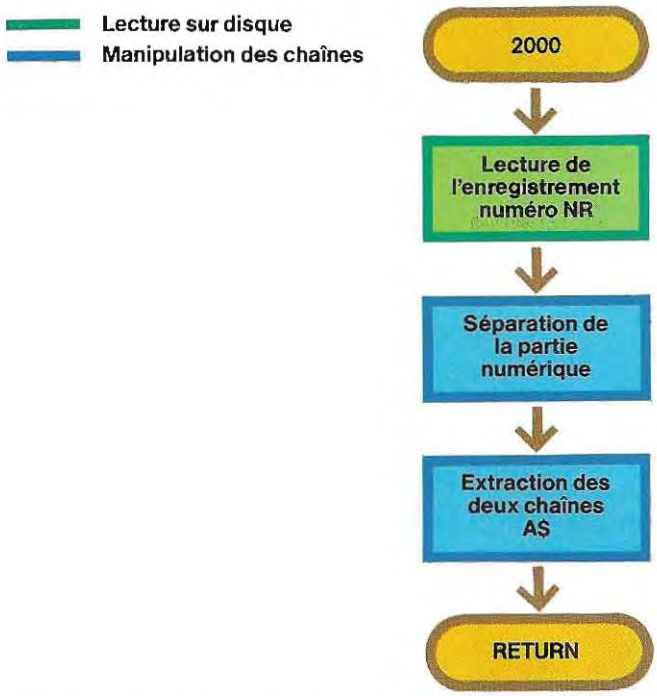
PREPARATION ET ECRITURE D'UN ENREGISTREMENT

Entrées du sous-programme : NR = Numéro de l'enregistrement.
V(4) = Valeurs numériques à écrire dans l'enregistrement NR.
LN = Longueur maximale (en nombre de chiffres) des valeurs numériques.
A$(2) = Chaînes à écrire dans l'enregistrement NR.
LS = Longueur maximale de chaque chaîne (en caractères).



La conversion des données peut aussi s'effectuer grâce aux fonctions STR$(R) et VAL(A$).
L'organigramme ci-dessus correspond à un sous-programme qui utilise ces fonctions pour préparer l'enregistrement à écrire sur disque. La page 657 illustre l'organigramme de la routine exécutant la lecture.
Les deux sous-programmes sont appelés par le programme principal, listé en pages 657 et 658. Pour faciliter la compréhension du sous-programme 2000, le schéma de la page 659 explique la logique mise en œuvre pour l'extraction des données.

LECTURE D'UN ENREGISTREMENT ET PREPARATION DES VALEURS NUMERIQUES



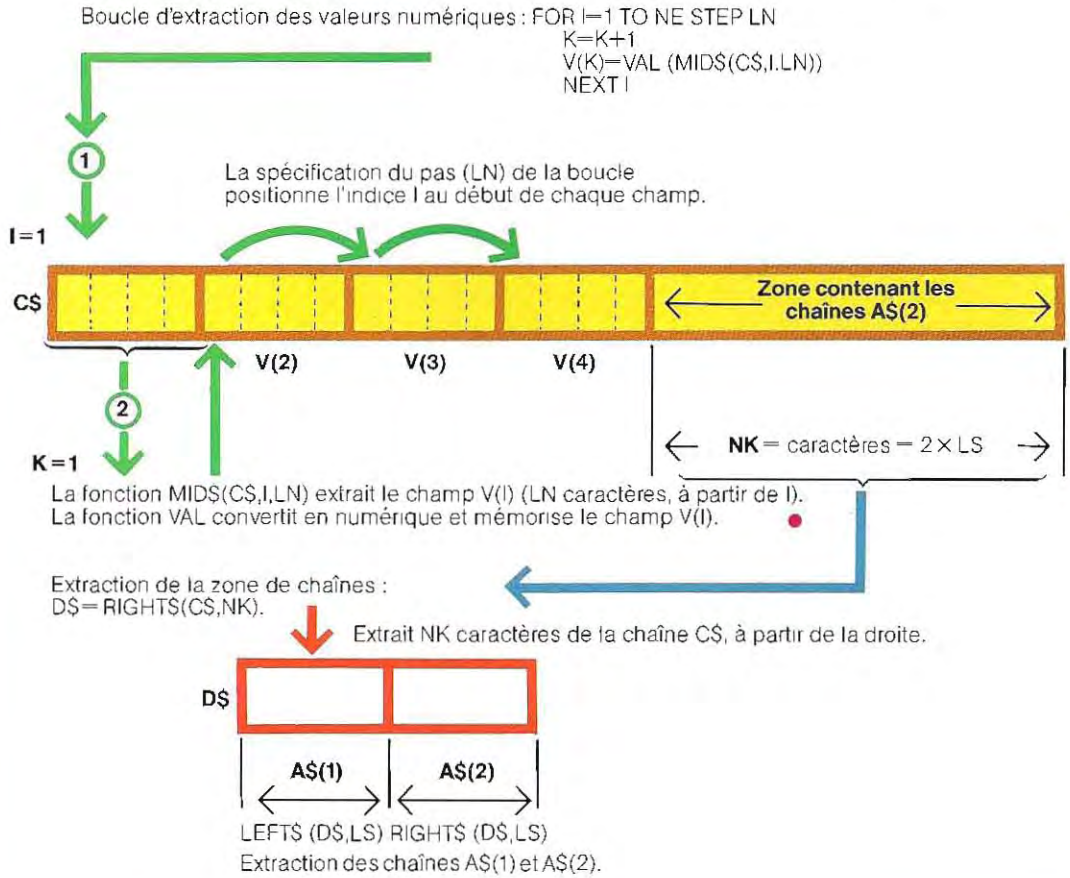
PREPARATION, LECTURE ET ECRITURE D'UN ENREGISTREMENT

```
10 '
20 '
30 '    PROGRAMME =LEC/EC
35 DEFINT I-N
40 ' **
50 '     NR = numero d'enregistrement a remplir
60 '     V(4) = valeurs numeriques a ecrire
70 '     LN = longueur max. des valeurs numeriques (nombre de chiffres)
80 '     A$(2) = chaines a ecrire a la suite des valeurs numeriques
90 '     LS = longueur max. des chaines (nombre de caracteres)
100 '
110 DIM B$(4), A$(2)
112 OPEN "R",1,"TEST2",50        'Les enregistrements ont 50 octets de longueur
114 FIELD #1,50 AS BUF$          'BUF$ est la memoire tampon associee au fichier TEST2
120 LS=10
130 LN=5
140 INPUT "Numero d'enregistrement a ecrire";NR
150 FOR I=1 TO 4
160 PRINT "Donnee numerique";I
170 INPUT "Valeur (max. 5 chiffres)";V(I)
180 NEXT I
190 INPUT "Premiere chaine";P$
200 INPUT "Seconde chaine";S$
202 A$(1)=SPACE$(LS) : A$(2)=SPACE$(LS)
204 LSET A$(1)=P$        'Transfert et cadrage a gauche des chaines
206 LSET A$(2)=S$        'lues
210 '
220 GOSUB 1000          'Preparation et ecriture de l'enregistrement
230 '
240 A$(1)="" : A$(2)=""      'Initialisations
250 FOR I=1 TO 5
252 V(I)=0
254 NEXT I
```

```
260 '
270 GOSUB 2000          ' Relecture et ecriture de l'enregistrement
280 '
290 LPRINT "Valeurs numeriques:"
300 FOR I=1 TO 4
310 LPRINT V(I)
320 NEXT I
330 LPRINT
340 LPRINT "Chaines:"
350 LPRINT A$(1)
360 LPRINT A$(2)
370 '
380 END
390 '
1000 ' ** SUBROUTINE D'ECRITURE SUR DISQUE **
1010 IF NR=0 THEN PRINT "Erreur sur le numero d'enregistrement" : GOTO 1240
1020 FOR I=1 TO 4
1030 B$(I)=SPACE$(LN)          ' Preparation des chaines d'espaces
1040 NEXT I
1050 FOR I=1 TO 4
1060 C$=STR$(V(I))             ' Conversion de V(I) en chaine
1070 IF LEN(C$)>LN THEN PRINT "Erreur sur la valeur numerique";I : GOTO 1240
1080 RSET B$(I)=C$             ' Transfert et cadrage a droite
1090 NEXT I
1100 C1$=A$(1)                 ' Transfert de la premiere chaine
1110 IF LEN(A$(1))>LS THEN C1$=LEFT$(A$(1),LS)
1120 C2$=A$(2)                 ' Transfert de la seconde chaine
1130 IF LEN(A$(2))>LS THEN C2$=LEFT$(A$(2),LS)
1140 ' *C1$ et C2$ contiennent les memes chaines que A$(1) et A$(2)
1150 ' mais eventuellement tronquees si elles etaient trop longues
1160 ' ** Preparation de la chaine a ecrire
1170 C$=""
1180 FOR I=1 TO 4
1190 C$=C$+B$(I)               ' Concatenation des chaines representant
1200 NEXT I                    ' les 4 valeurs numeriques
1210 C$=C$+C1$+C2$
1212 PRINT "Les donnees sont stockees dans l'enregistrement, sous la forme:"
1214 PRINT C$
1220 BUF$=C$                   ' Transfert dans la memoire tampon
1230 PUT £1,NR                 ' Ecriture sur disque
1240 RETURN
1250 '
2000 ' ** SUBROUTINE DE LECTURE SUR DISQUE **
2010 GET £1,NR                 ' Lecture
2020 C$=BUF$                   ' C$ = buffer intermediaire, contenant les donnees
2025 '                          telles qu'elles sont lues
2030 PRINT "Enregistrement lu";C$
2040 NE=4*LN                   ' Longueur en octets de la zone numerique,
2050 '                          soit 4 champs de longueur LN
2060 K=0
2070 FOR I=1 TO NE STEP LN
2080 K=K+1
2090 V(K)=VAL(MID$(C$,I,LN))
2100 NEXT I
2110 NK=2*LS                   ' Longueur de la zone alphanumerique
2120 ZA=RIGHT$(C$,NK)          ' Extraction de cette zone qui contient
2130 '                          les deux chaines
2140 A$(1)=LEFT(ZA$,LS)        ' Separation des deux chaines
2150 A$(2)=RIGHT(ZA$,LS)
2160 RETURN
```

```
Valeurs numeriques:
 1111
 2222
 3333
 4444

Chaines:
AAAAAAAAAA
BBBBBBBBBB
```

SCHEMAS LOGIQUES D'EXTRACTION DES DONNEES



Sélection des données

Au cours de la gestion des fichiers, il arrive rarement d'avoir à traiter la totalité des enregistrements existants. En règle générale, les mises à jour, recherches ou autres traitements n'intéressent que certains enregistrements, reconnaissables par les valeurs de certains champs déterminés.

Ces champs constituent la **clé** d'accès à l'enregistrement et, lorsque le système le permet, le programmeur préfèrera adopter une structure de fichier indexé. De cette manière, toute la gestion des pointeurs est assurée par le système, et l'utilisateur n'a à sa charge, dans son programme, que quelques instructions spécifiant la clé qu'il entend utiliser pour sélectionner l'enregistrement.

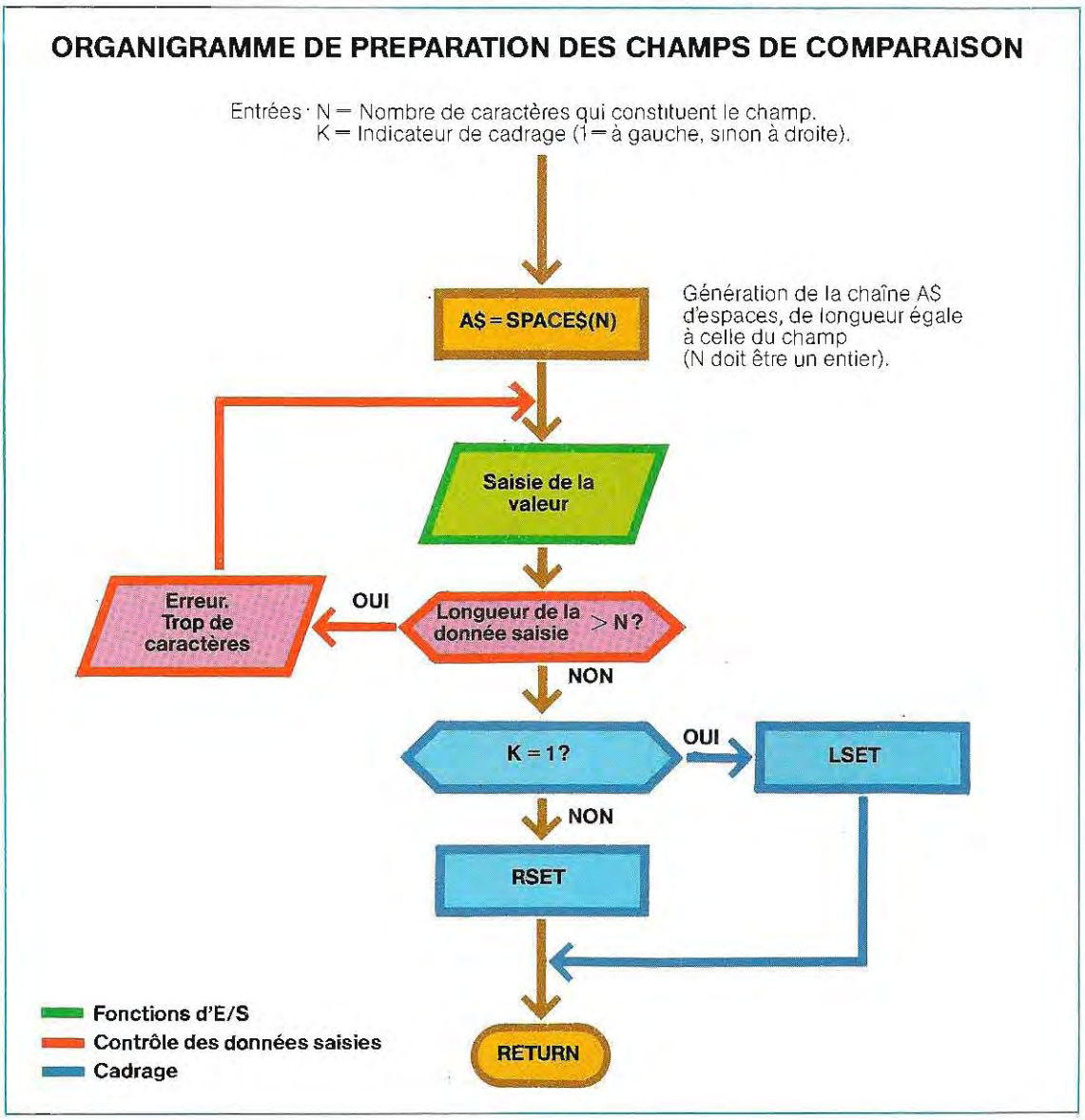
Le logiciel de base (bibliothèque de programmes « utilitaires ») permettant ce type de gestion n'existe pas sur tous les systèmes, et très souvent, le programmeur devra écrire lui-même ses « utilitaires ».

Prenons l'exemple d'une routine de recherche d'un enregistrement selon une valeur de champ donnée. Une programmation de ce genre ne présente pas de difficultés particulières : il s'agit uniquement d'utiliser les opérateurs logiques à bon escient. La seule précaution à prendre est de justifier (cadrer) tous les champs de la même manière.

En général, lors de l'écriture sur disque, les champs ne sont pas complètement remplis de caractères, et il demeure des blancs. Lorsque le programme lit sur disque en enregistrement, tous les caractères sont chargés en mémoire, y compris les blancs. Ainsi, un champ extrait d'un fichier et un autre saisi au clavier seront peut-être cadrés différemment, et une comparaison entre les deux donnera un résultat négatif, même s'ils contiennent les

mêmes caractères (exceptés les blancs).
Supposons, par exemple, qu'un fichier mémorisé sur disque comporte un champ pour la saisie d'un nom de ville, d'une longueur de 10 caractères : la donnée tapée au clavier en phase de saisie est LYON. En ce cas, le champ contient, à la suite des quatre caractères de la donnée, six blancs. Afin de rechercher dans le fichier l'enregistrement contenant la donnée LYON, il faut préparer une chaîne de comparaison possédant les mêmes caractéristiques : la donnée doit être cadrée à gauche, et l'espace restant doit être laissé en blanc.
La convention habituelle consiste à cadrer les chaînes de caractères à gauche et les chaînes représentant des valeurs numériques à droite ; on peut bien sûr adopter une convention différente, à condition de la respecter pour toutes les opérations de fichier.
Dans l'exemple précédent, il suffit de préparer une chaîne d'espaces, de la longueur du champ, et d'y transférer, correctement cadrée selon la valeur de l'indicateur K, la donnée saisie au clavier. La chaîne ainsi construite pourra être utilisée comme valeur de comparaison, dans une boucle de lecture et de sélection des enregistrements.
L'organigramme ci-dessous illustre schématiquement la préparation de cette chaîne.

ORGANIGRAMME DE PREPARATION DES CHAMPS DE COMPARAISON



Commandes et fonctions spéciales

Les systèmes d'exploitation limitent toujours le nombre de fichiers qui peuvent être ouverts simultanément (par exemple, trois sous CP/M) ; d'éventuelles modifications de cette limite doivent être explicitement communiquées au système lors du chargement de l'interpréteur BASIC.
Dans ce cas, le format de commande du chargement de l'interpréteur est habituellement le suivant :

MBASIC/F:N

Le symbole / sert de délimiteur entre la commande MBASIC et les options choisies ; la lettre F indique qu'on veut ouvrir plus de fichiers que la limite standard, et la valeur N représente le nombre de fichiers désiré.
Ainsi, la commande

MBASIC/F:5

prépare l'interpréteur à manipuler simultanément cinq fichiers. Sous le système d'exploitation CP/M, l'espace mémoire représente 166 octets pour chaque fichier. Dans l'exemple précédemment considéré, la zone mémoire réservée aux blocs de description (FDB : File Description Block) des cinq fichiers serait de : $5 \times 166 = 830$ octets.
Une autre option est possible :

/ S:N

Le nombre N indique la longueur maximale des enregistrements, exprimée en octets.
Si elle n'est pas spécifiée, cette longueur vaut, par défaut, 128 octets (la spécification d'une longueur maximale n'empêche pas d'attribuer des longueurs plus petites aux buffers dans l'instruction FIELD).
Rappelons que, sous CP/M, le nombre maximum d'enregistrements logiques adressables se limite à 32767, sauf si l'on définit, au moyen de l'option S, une longueur d'enregistrements de 256 octets ; en ce cas, l'adressage maximal est de 8 Méga-octets.
Sur certains interpréteurs BASIC, une fonction spéciale permet de connaître la taille des fichiers (en secteurs).
En Basic 80, cette fonction est :

LOF(N)

qui délivre le nombre d'enregistrements de la dernière extension du fichier de numéro logique N. Si ce fichier ne possède qu'une seule extension, ce nombre coïncide avec sa taille réelle.

Fichiers multivolumes

La capacité de stockage des disquettes varie de 160 Kilo-octets à 1 Méga-octet, selon leurs dimensions et leurs types. Pour certaines applications, il s'avère parfois nécessaire de disposer d'une capacité plus grande que celle offerte par la disquette.
La meilleure solution consiste évidemment à utiliser des supports mieux adaptés, comme, par exemple, les Winchester (qui sont des disques durs). Sur les micro-ordinateurs, certains permettent de stocker jusqu'à 36 Méga-octets, les modèles les plus usuels restant entre 5 et 10 Méga-octets. En outre, les disques durs offrent l'avantage de temps d'accès très inférieurs lors des fonctions d'E/S.
Cependant, des questions de coût ne permettent pas toujours d'adopter cette solution, optimale d'un point de vue technique. Il faut alors, par programme, suppléer aux limites du matériel.
La méthode classique consiste à partager le fichier en plusieurs sections, qui seront stockées séparément sur disquettes.
Le programme d'application devra, bien sûr, tenir compte de cette répartition, de manière à guider l'utilisateur dans le choix de la disquette appropriée. Au moment du traitement, le programme déterminera la disquette sur laquelle sont stockées les données en cours, en informera l'utilisateur, et attendra que le montage de la disquette requise soit achevé.
Un tel fichier, caractérisé par sa subdivision en plusieurs parties, est dit **multivolume**, et chacune de ses parties est appelée **volume**.
Cette répartition des données qui, sur les micro-ordinateurs ou les ordinateurs personnels, permet souvent d'accroître les capacités de stockage, peut aussi se justifier par un choix logique. Il n'est pas toujours souhaitable de mémoriser des données dans un fichier unique ; parfois, il s'avère plus avantageux de diviser le fichier en plusieurs volumes, même si le disque est capable de le contenir en entier. Dans ce cas, on opère une division de

type logique : la répartition en volumes existe, mais tous les volumes résident sur la même disquette. Le fichier étant structuré de cette façon, le programme d'application qui l'utilise n'a pas à attendre le montage des différents disques.
Un programme de gestion de fichiers multivolumes sera extrêmement complexe s'il opère sur des fichiers indexés, alors qu'il s'avère relativement simple dans le cas de fichiers en accès direct. La logique de gestion d'un fichier en accès direct repose sur l'emploi du numéro d'enregistrement comme clé d'accès aux données : la première donnée saisie portera le numéro d'enregistrement 1, la deuxième sur le numéro 2 et ainsi de suite. Rappelons, toutefois, qu'assimiler le numéro d'enregistrement à une clé d'accès est un abus de langage ; l'adresse physique de chaque enregistrement se calcule, en réalité, en soumettant les données à des traitements relativement complexes. La gestion d'un fichier multivolume consiste à mémoriser dans un fichier à part, les noms des volumes (c'est-à-dire des disquettes) et pour chacun, les numéros de premier et de dernier enregistrement stockés. Ce fichier, qui retrace l'évolution du fichier multivolume, constitue une sorte de répertoire, et ne comporte que peu d'enregistrements (un par disquette) constamment mis à jour.

Gestion des fichiers sur les grands systèmes

Sur les systèmes plus complexes, à partir des mini-ordinateurs, sont implantés des programmes « utilitaires » de gestion des bases de données, qui déchargent le programmeur d'une grande partie des tâches de détail.
Ces programmes de gestion se rangent en deux catégories : programmes spécialisés autonomes et programmes nécessitant un langage hôte.
L'ensemble des programmes spécialisés constitue un véritable langage de haut niveau, comportant des « instructions » comparables à celles d'un langage de programmation, ainsi que des commandes spécifiques à la gestion des fichiers. Les programmes rattachés à un langage hôte sont de véritables routines formées d'une série d'instructions, et qui accroissent les possibilités du langage.

Sur les gros systèmes, les principaux langages hôtes (toujours compilés) sont le Cobol et le Fortran ; les commandes inhérentes à la gestion des fichiers sont traduites sous une forme compatible avec le langage hôte (lors de la compilation), puis exécutées.
Les commandes fondamentales prévues dans tout système d'exploitation sont :

STORE écrit un enregistrement, et crée toutes les chaînes éventuelles de pointeurs vers les autres données ;
FIND recherche un ensemble de données en fonction de paramètres fournis par l'utilisateur ;
MODIFY modifie une ou plusieurs données ;
ERASE efface un enregistrement, et modifie en conséquence les chaînes de pointeurs qui lui sont associées.

De nombreux langages élaborés spécifiquement pour la gestion des bases de données sont pourvus de « sous-ensembles » d'interrogation, véritables procédures de recherche de données, accessibles par un personnel qui n'a pas nécessairement des connaissances de programmation. La syntaxe des instructions est très simple et tente de se rapprocher des phrases en langage naturel. Par exemple, l'interrogation d'un fichier de produits, en utilisant le système MDQS (Management Data Query System) ressemblera presque à un dialogue :

```
DISPLAY PRODUIT TYPE QUANTITE
IF CODE = "A"
LET QUANTITE = ENTREE-SORTIE
END
```

L'emploi d'un tel langage de commandes ne requiert que la connaissance de quelques mots anglais, faciles à retenir : DISPLAY = montre, affiche ; IF = si ; LET = soit, etc.

La gestion d'une entreprise

La gestion d'une petite entreprise constitue l'une des applications typiques d'un ordinateur, dans la mesure où les fonctions à exécuter sont répétitives et impliquent de nombreuses données.
La conception d'un logiciel de gestion présente des difficultés non négligeables, dues à la nécessité de relier les données de nom-

breuses procédures successives et à l'exécution de fonctions spécifiques d'ordre économique ou fiscal.
Pour les applications d'une certaine envergure, il n'est pas avantageux d'écrire son propre logiciel : on choisira l'un des nombreux systèmes disponibles sur le marché, en l'adaptant éventuellement selon ses besoins.
Etant donné la complexité du sujet, on ne va exposer ici que les problèmes majeurs associés à la gestion d'une entreprise.
Les principales questions qu'un logiciel de gestion doit résoudre peuvent s'énumérer de la façon suivante :

TEST 19

1 / Parmi ces affirmations, lesquelles sont vraies, lesquelles sont fausses ?

Un fichier séquentiel :
a) Ne permet pas l'ajout d'enregistrements.
b) Possède une vitesse d'accès élevée.
c) Occupe moins de place qu'un fichier en accès direct.
Un fichier en accès direct :
d) Ne peut posséder d'extensions.
e) Peut être utilisé indifféremment en lecture ou en écriture.
f) Dispose de la fonction EOF(N).

---

2 / Que provoque l'exécution des instructions suivantes, si le fichier ESSAI n'existe pas sur l'unité de disque A ?

a) OPEN"I",1,"A:ESSAI"
b) OPEN"R",1,"A:ESSAI",120

---

3 / Au moyen de quelle instruction ou fonction peut-on déterminer le numéro du dernier enregistrement utilisé après une instruction PUT ou GET ?

---

4 / Ecrire une routine de saisie des données suivantes :

C% = code numérique (entier compris de 1 à 99).
D$ = description (20 caractères).
QT% = quantité (nombre entier).
C = coût.

Puis d'écriture dans un fichier en accès direct nommé ESSAI, résidant sur le disque monté sur l'unité A.

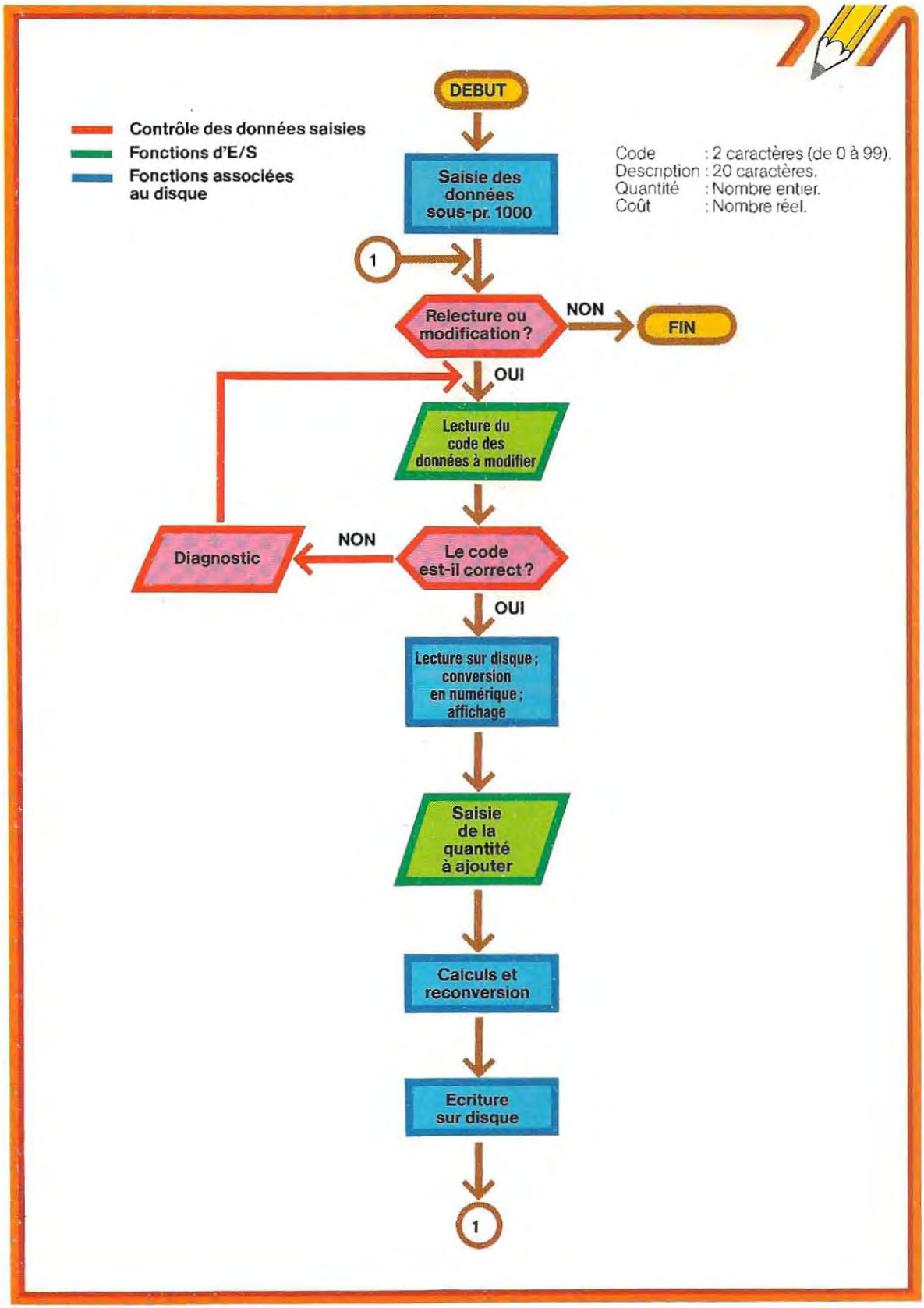
---

5 / Compléter la routine précédente par un programme principal contenant les instructions pour :

- relire une donnée quelconque du fichier ESSAI ;
- ajouter à la quantité existante QT%, une nouvelle quantité, saisie au clavier ;
- enregistrer la modification sur disque (à la même position).

L'organigramme de la routine est illustré page 664.

Les solutions du test se trouvent page 671.


Contrôle des données saisies
Fonctions d'E/S
Fonctions associées au disque
Code : 2 caractères (de 0 à 99).
Description : 20 caractères.
Quantité : Nombre entier.
Coût : Nombre réel.
DEBUT
Saisie des données sous-pr. 1000
1
Relecture ou modification ?
NON
FIN
OUI
Lecture du code des données à modifier
Le code est-il correct ?
NON
Diagnostic
OUI
Lecture sur disque ; conversion en numérique ; affichage
Saisie de la quantité à ajouter
Calculs et reconversion
Ecriture sur disque
1

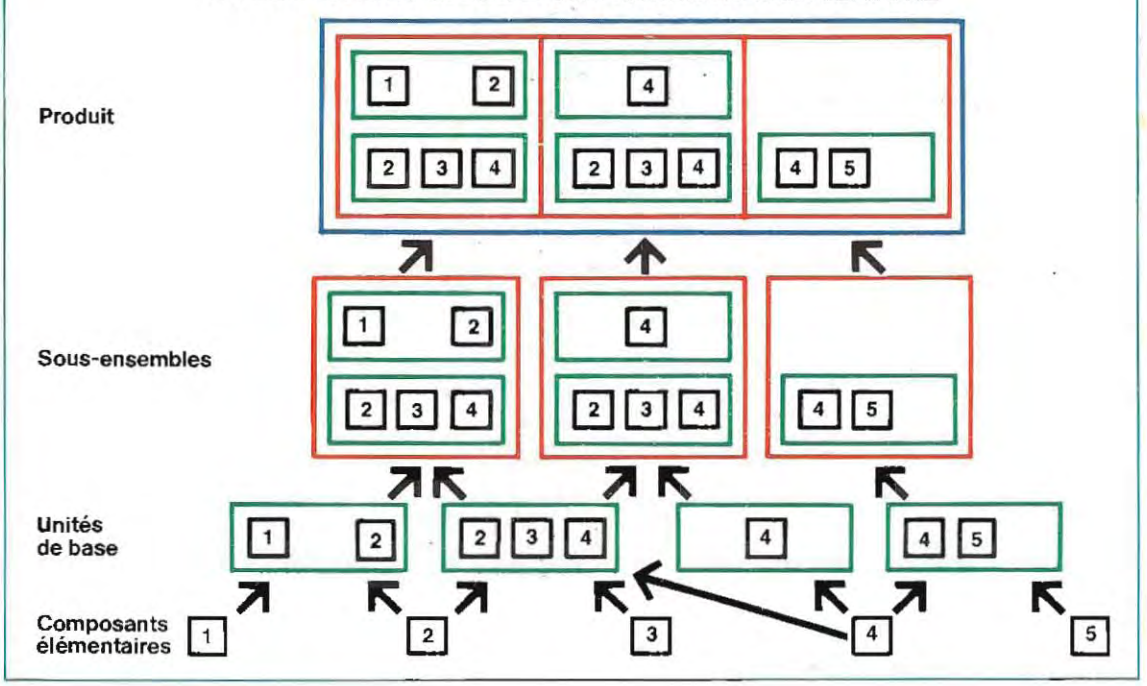
Réévaluation des stocks et inventaires
Fichier Nomenclature
Simulation
Relations entre procédures.

Réévaluation des stocks et inventaires. L'entreprise reçoit ses marchandises (matières premières) selon un calendrier préétabli ; le coût d'une marchandise donnée varie donc en fonction de sa date de réception. Au moment de la vente d'un produit fini, il s'agit de trouver un moyen d'établir son prix de revient, qui tiendra compte, entre autres, du coût de la matière première, et donc de sa date d'arrivée. Il faut donc réserver de l'espace mémoire pour stocker les dates de réception de toutes les marchandises. En outre, si le logiciel de gestion doit satisfaire à des normes d'ordre fiscal, la logique d'attribution de la valeur du produit en fonction de sa date d'arrivée ne peut être arbitraire.

Fichier Nomenclature. La gestion d'une entreprise de production s'appuie généralement sur un fichier Nomenclature, c'est-à-dire le répertoire des composants nécessaires à la fabrication du produit.
En général, un produit est formé de plusieurs sous-ensembles, chacun d'eux constitué également de plusieurs unités et chacune d'elles de plusieurs composants. On crée ainsi une échelle hiérarchisée spécifiant, pour chaque produit, le nombre et le type d'éléments nécessaires au niveau inférieur.
On comprendra aisément que les liens d'inter dépendance ainsi créés engendrent des difficultés notables en ce qui concerne l'écriture des programmes. Le schéma ci-dessous représente une structure à quatre niveaux. En observant les différentes couleurs, on peut se faire une idée de la complexité des pointeurs à gérer pour maintenir l'intégrité de cette structure.

Simulation. A partir du fichier Nomenclature, on peut réaliser des traitements particuliers, qui vont simuler l'effet produit par des variations de coûts, ou par le manque de quantité en stock d'un ou plusieurs composants. En suivant les liens de dépendance logiques, le logiciel de simulation est en mesure de déterminer sur quels produits, et dans quelles proportions ces différentes causes vont se répercuter.
La simulation d'événements préétablis, et l'étude de l'effet de leur propagation sur la structure hiérarchisée constituent un excellent outil de choix stratégiques d'une entreprise. L'importance de ces méthodes est telle

STRUCTURE LOGIQUE D'UN FICHIER NOMENCLATURE



que plusieurs créateurs de logiciels proposent actuellement des programmes de simulation de portée générale et pouvant être implantés sur micro-ordinateurs.

Relations entre procédures. La gestion d'une entreprise, même de dimensions modestes, implique un échange constant de données de provenances diverses. Plus cet échange est automatisé et rapide, plus la gestion de la structure dans son entier s'en trouve facilitée.
On comprend aisément que, pour une bonne gestion, tous les niveaux doivent interagir (la détermination des coûts, les modes de livraison, les bilans, etc.). En d'autres termes, les différents services doivent pouvoir accéder à des données communes. Dans les entreprises de grande dimension, cet aspect de la question est résolu par l'installation d'un site central qui contient les fichiers généraux. Même dans ce cas, quelques problèmes subsistent parfois : l'utilisateur final ne possède pas toujours les connaissances spécialisées qui lui offriraient un accès aisé aux procédures générales et, très souvent, il préfère développer ses propres traitements particuliers, qui ne s'intègrent pas dans la bibliothèque de programmes généraux.
Les micro-ordinateurs peuvent résoudre aisément ces problèmes, en offrant une solution facilement intégrable au travail de bureau.
L'emploi d'un logiciel « clés en main » par l'utilisateur final ne requiert pas de connaissances particulières et peut être appris en quelques jours. En outre, grâce à des réseaux appropriés, les différents postes de travail peuvent dialoguer entre eux ou avec le site central, et échanger des informations en temps réel.
L'évolution prévisible de la fonction de programmeur sur ces machines configurées en réseau sera, dans un futur proche, celle d'ingénieur-système : maîtrisant la connaissance des capacités du matériel et des possibilités des différents logiciels d'application, il sera en mesure de conseiller les utilisateurs dans leur choix et leurs installations.

Unité de disque dans un centre de calcul.

J. Pickerell/Marka

Exemple d'application : la gestion d'un répertoire

Afin de résumer les concepts exposés au sujet de la gestion des fichiers, nous voudrions maintenant décrire la méthode de création d'un programme de gestion d'une rubrique. Quel que soit le fichier à gérer, et quelle que soit l'application, les fonctions à exécuter sont les suivantes :

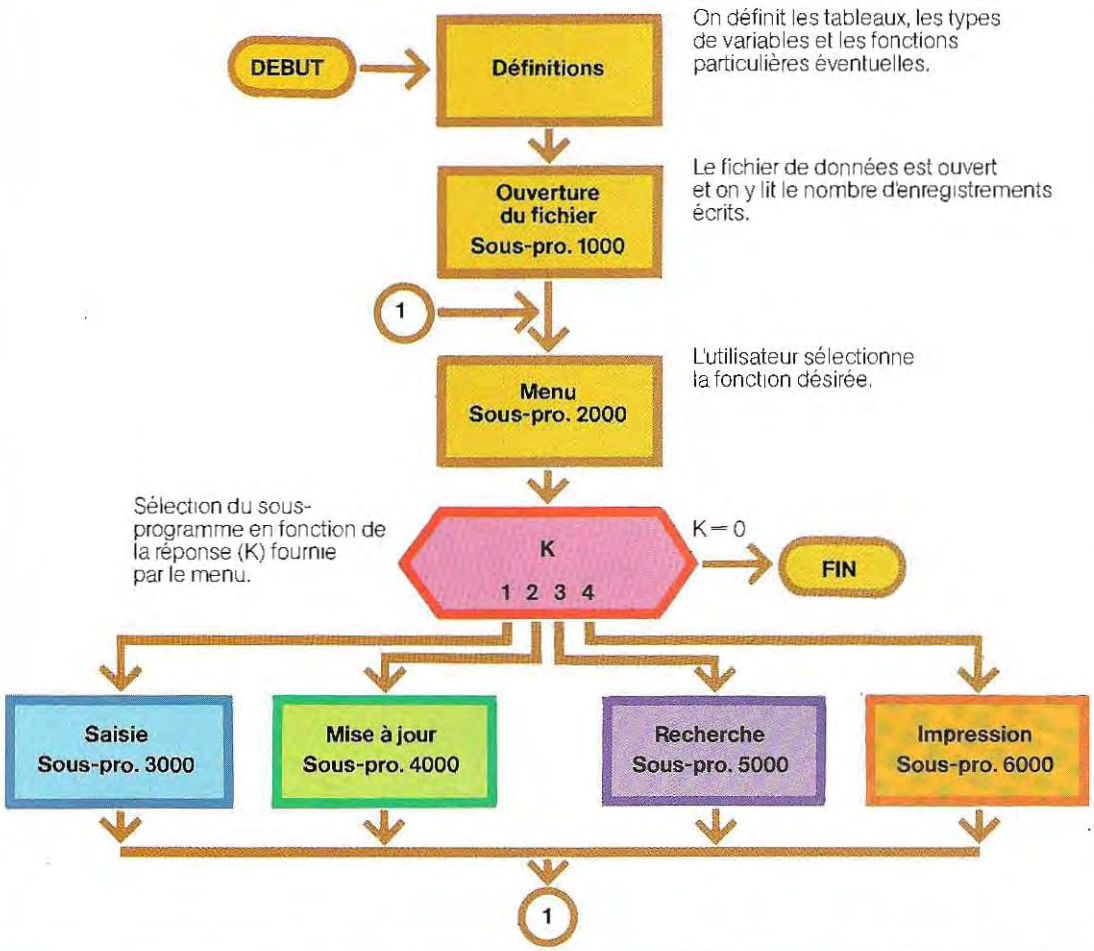
- saisie des nouvelles données ;
- modification des données existantes ;
- recherche d'une donnée en fonction de certains paramètres ;
- impression.

Ces quatre fonctions seront appelées par un menu, dans le programme principal de gestion du fichier, et seront exécutées par autant de sous-programmes.
La page 667 illustre l'organigramme général : le programme principal ne contient que des définitions et l'appel aux sous-programmes ; toutes les fonctions du programme sont exécutées dans les sous-programmes.

Ouverture du fichier (sous-programme 1000)

Ce sous-programme ouvre le fichier, définit les champs et lit le numéro du dernier enregistrement rempli. Le fichier de taille quelconque peut en effet n'être occupé qu'en partie.

GESTION D'UN FICHIER SUR DISQUE. ORGANIGRAMME GENERAL



Pour écrire un nouvel enregistrement, il faut connaître le numéro du dernier enregistrement rempli ; on incrémente de 1 cette valeur. On crée un fichier de la taille maximale prévue et on utilise son dernier enregistrement pour y stocker le nombre d'enregistrements déjà remplis. Ce dernier enregistrement devra être mis à jour au fur et à mesure que de nouvelles données seront entrées (voir p. 668).
Dans notre application, le fichier contiendra les champs suivants :

Nom	= 20 caractères
Rue	= 20 caractères
Téléphone	= 12 caractères
Ville	= 20 caractères
Longueur d'un enregistrement	= 72 caractères

La création du fichier s'effectue de façon immédiate, en entrant au clavier l'instruction :

```
OPEN"R",1,"A:DATA",72
```

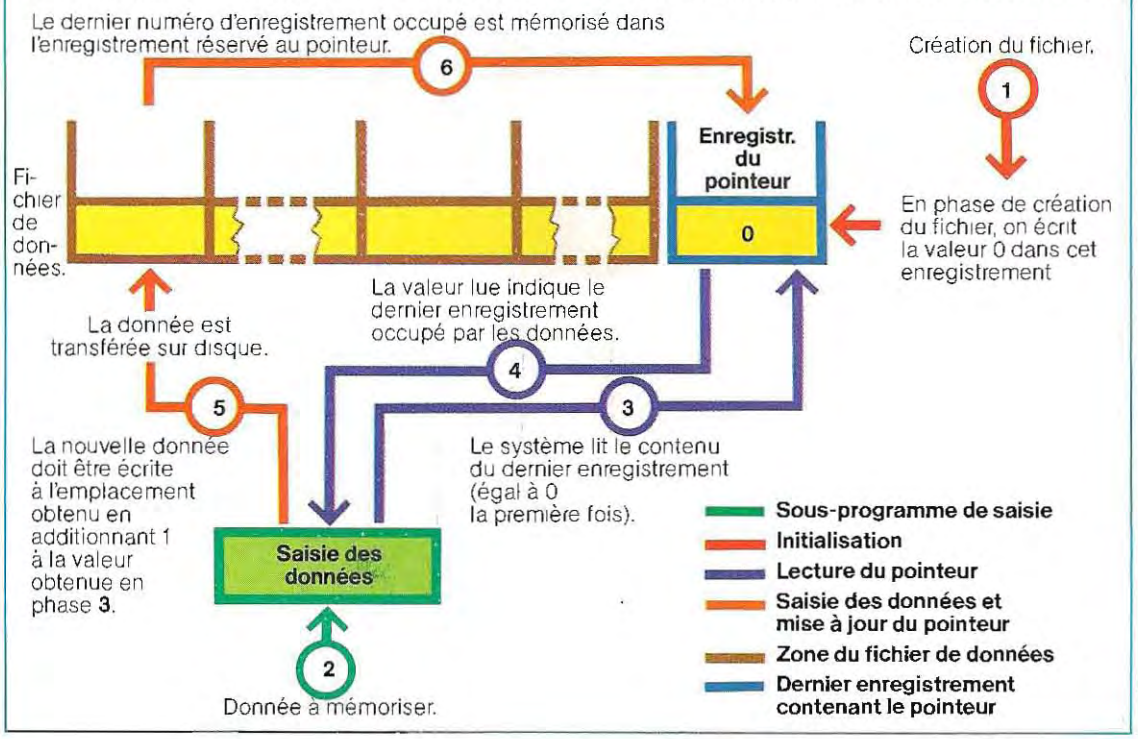
Le système mémorise alors le nom du fichier dans le répertoire du disque A et lui attribue une taille initiale. Si nous avons au maximum 100 groupes de données à stocker, il faut écrire dans l'enregistrement 101 (virtuellement le dernier) la valeur 0 : au début, le fichier est vide et le premier enregistrement disponible porte le numéro 1 (0+1=1).
La création du fichier et l'initialisation du dernier enregistrement peuvent être réalisées par un programme utilitaire, de portée générale, dont le listing est illustré à la page 668, en bas.

Une fois que la procédure de création et d'initialisation est réalisée, on peut passer à l'écriture du sous-programme 1000, qui devra exécuter les fonctions suivantes :

- ouverture du fichier ;
- lecture du dernier enregistrement.

La structure de ce sous-programme est très simple ; le listing correspondant est reproduit en haut de la page 669.
Comme on le voit, cette routine est paramétrée, et tous les paramètres lui sont transmis par le programme principal. De cette façon, elle peut être utilisée par n'importe quelle application.

GESTION DES FICHIERS. UTILISATION DU POINTEUR SUR LE DERNIER ENREGISTR.



CREATION DU FICHIER ET INITIALISATION DU DERNIER ENREGISTREMENT

```
10 ' ** Programme utilitaire de creation du fichier
20 '    et d'initialisation du pointeur (dernier enregistrement) **
30 '
40 '       PROGRAMME CREAT
50 INPUT "NOM DU FICHIER ";NOM$
60 INPUT "SUR QUELLE UNITE DISQUE (A:/B:) ";D$
70 IF D$<>"A:" AND D$<>"B:" GOTO 60
80 NM$=D$+NOM$
90 INPUT "Longueur des enregistrements (Nb de caracteres) ";L
100 IF L>128 GOTO 90
110 OPEN "R",1,NM$,L
120 FIELD 1,L AS BF$
130 INPUT "Nombre d'enregistrements maximum ";NE
140 MX=NE+1          ' Numero du dernier enregistrement
150 B$="0" : C$=STRING$(L,B$)       ' C$="000...0"
160 BF$=C$
170 PUT 1,MX         ' Ecriture pointeur
180 PRINT "Le fichier ";NM$;" est créé"
190 PRINT "Longueur=";NE
200 PRINT "Pointeur en position=";MX
210 END
```

ROUTINE D'OUVERTURE DU FICHIER

```
1000 ' ** Subroutine d'ouverture du fichier **
1010 '
1020 '    Entrees =
1030 '       NM$ = nom complet du fichier (A : DONNEE par exemple)
1040 '         L = longueur des enregistrements
1050 '        NE = longueur du fichier (nombre d'enregistrements)
1060 '       BF$ = buffer associe au fichier
1070 '
1080 OPEN "I",1,NM$ : CLOSE 1       'Verification de l'existence du fichier
1090 OPEN "R",1,NM$,L
1100 FIELD 1,LMS BF$
1110 MX=NE+1
1120 GET 1,MX              'Lecture du pointeur
1130 A$=BF$                'La chaine est affectee a A$
1140 MAX=VAL(A$)           'Conversion en numerique
1150 '
1160 'En sortie, MAX contient la valeur stockee dans le dernier
1170 'enregistrement du fichier.
1180 'C'est le numero du dernier enregistrement occupe
1190 'Faire +1 pour obtenir le numero du premier enregistrement libre
1200 '
1210 RETURN
```

ROUTINE DE SAISIE DES DONNEES

```
3000 ' ** SAISIE DES DONNEES **
3010 '    Entrees =
3020 '       MAX = numero du dernier enregistrement occupe
3030 '        NE = longueur du fichier
3035 '       BF$ = buffer associe au fichier
3040 'Les instructions de saisie sont propres a cette application
3042 'Il faudra eventuellement les modifier en fonction
3044 'des exigences particulieres du fichier traite
3050 ' * Saisie des donnees au clavier
3060 INPUT "NOM ";N$
3070 INPUT "RUE ";R$
3080 INPUT "TELEPHONE ";T$
3090 INPUT "VILLE ";V$
3100 ' * Preparation de l'enregistrement
3110 A1$=SPACE$(20)        'Creation des quatre chaines d'espaces
3120 A2$=SPACE$(20)
3130 A3$=SPACE$(12)
3140 A4$=SPACE$(20)
3150 LSET A1$=N$           'Cadrage a gauche des donnees
3160 LSET A2$=R$           'dans les champs d'espaces,
3170 LSET A3$=T$           'afin de faciliter des comparaisons ultérieures
3180 LSET A4$=V$
3190 'L'instruction LSET tronque les caracteres excedentaires
3192 '
3200 A$=A1$+A2$+A3$+A4$        'Concatenation
3210 BF$=A$
3220 M=MAX+1                   'Premier enregistrement libre
3230 PUT 1,M                   'Ecriture
3240 IF MODE=1 GOTO 3310       'Si MODE=1, on vient de la routine 4000
3250 '                          donc on ne modifie pas le pointeur
3255 '
3260 MAX=MAX+1                 'Incrementation du pointeur
3270 B$=STR$(MAX)              'Conversion en chaine
3280 A$=SPACE$(L-LEN(B$))      'Creation d'une chaine d'espaces
3290 BF$=A$+B$                 'pour completer l'enregistrement
3295 MX=NE+1                   'Numero du dernier enregistrement
3300 PUT 1,MX
3310 RETURN
```

Insistons sur le fait que l'instruction 1090 (ouverture en mode « I » et fermeture du fichier) a seulement pour but de vérifier l'existence du fichier : si l'on ouvre en mode « R » un fichier non créé, on en crée un nouveau non initialisé.

Menu (sous-programme 2000)

En ce qui concerne ce sous-programme, le lecteur se référera à l'un de ceux déjà présentés lorsque nous avons abordé le sujet d'un point de vue général. L'unique changement réside dans les messages à prévoir.

Saisie (sous-programme 3000)

Les fonctions à exécuter sont les suivantes :
– lecture des données depuis la console ;
– leur écriture sur le premier enregistrement disponible ;
– mise à jour du dernier enregistrement afin de prendre en compte la nouvelle donnée saisie.
Ce sous-programme est listé en bas de la page 669.

Mise à jour (sous-programme 4000)

Elle doit exécuter les fonctions suivantes :
– lecture, à partir du clavier, du numéro d'enregistrement à mettre à jour ;
– lecture des données concernées, depuis le disque ;
– affichage de ces données ;
– lecture des nouvelles valeurs (modification des données existantes) ;
– écriture des nouvelles données dans le même enregistrement.
Le listing correspondant est illustré ci-dessous. Pour le programmer, on peut utiliser en partie la routine de saisie.
La différence entre les phases de saisie et de mise à jour réside dans l'emplacement différent où sont écrites les données : durant la saisie, chaque donnée est placée dans un nouvel enregistrement, alors qu'en mise à jour, la donnée doit être récrite sur le même enregistrement, et le pointeur de fin de fichier (numéro du dernier enregistrement) ne doit pas être incrémenté.
Le sous-programme 4000 pourrait donc directement appeler la routine 3000, après avoir positionné un indicateur (sur le listing, KS) indiquant au sous-programme 3000 de ne pas mettre à jour le pointeur sur les données.
Il faut également modifier le pointeur (MAX) qui désigne l'enregistrement à remplir.
On pourrait également récrire directement dans le sous-programme 4000 les instructions nécessaires (lignes 3050 à 3230). L'accroissement de longueur du programme est négligeable et on y gagne en clarté.

ROUTINE DE MISE A JOUR

```
4000 ' ** MISE A JOUR **
4005 '
4010 INPUT "Numero d'enregistrement a modifier";N
4020 IF N>NE GOTO 4010      ' Erreur
4030 GET 1,N
4040 A$=BF$
4050 ' * Extraction des champs
4060 N$=LEFT$(A$,20)
4070 R$=MID$(A$,21,20)
4080 T$=MID$(A$,41,12)
4090 U$=RIGHT$(A$,20)
4100 PRINT "DONNEES LUES DANS L'ENREGISTREMENT ";N
4110 PRINT C$:PRINT R$
4120 PRINT T$:PRINT U$
4130 ' * Pour la saisie des nouvelles donnees,
4140 '   on appelle la subroutine 3000 de saisie
4150 '   on positionne un Flag (MODE) a 1 pour que
4160 '   les modifications ne soit pas considerees comme une nouvelle donnee
4170 NS=MAX          ' Sauvegarde du nombre d'enregistrements
4180 MAX=N-1         ' Prepare la position d'ecriture
4190 MODE=1
4200 GOSUB 3000
4210 MAX=NS          ' On restaure la valeur du pointeur
4220 RETURN
```

Solutions du test 19

1 / a) est vrai en précisant ceci : les ajouts sont possibles au moyen d'un fichier intermédiaire (et non directement) ;
b), c), d) sont faux ; e) vrai ; f) faux : la fonction EOF(N) est employée pour indiquer la fin d'un fichier séquentiel.

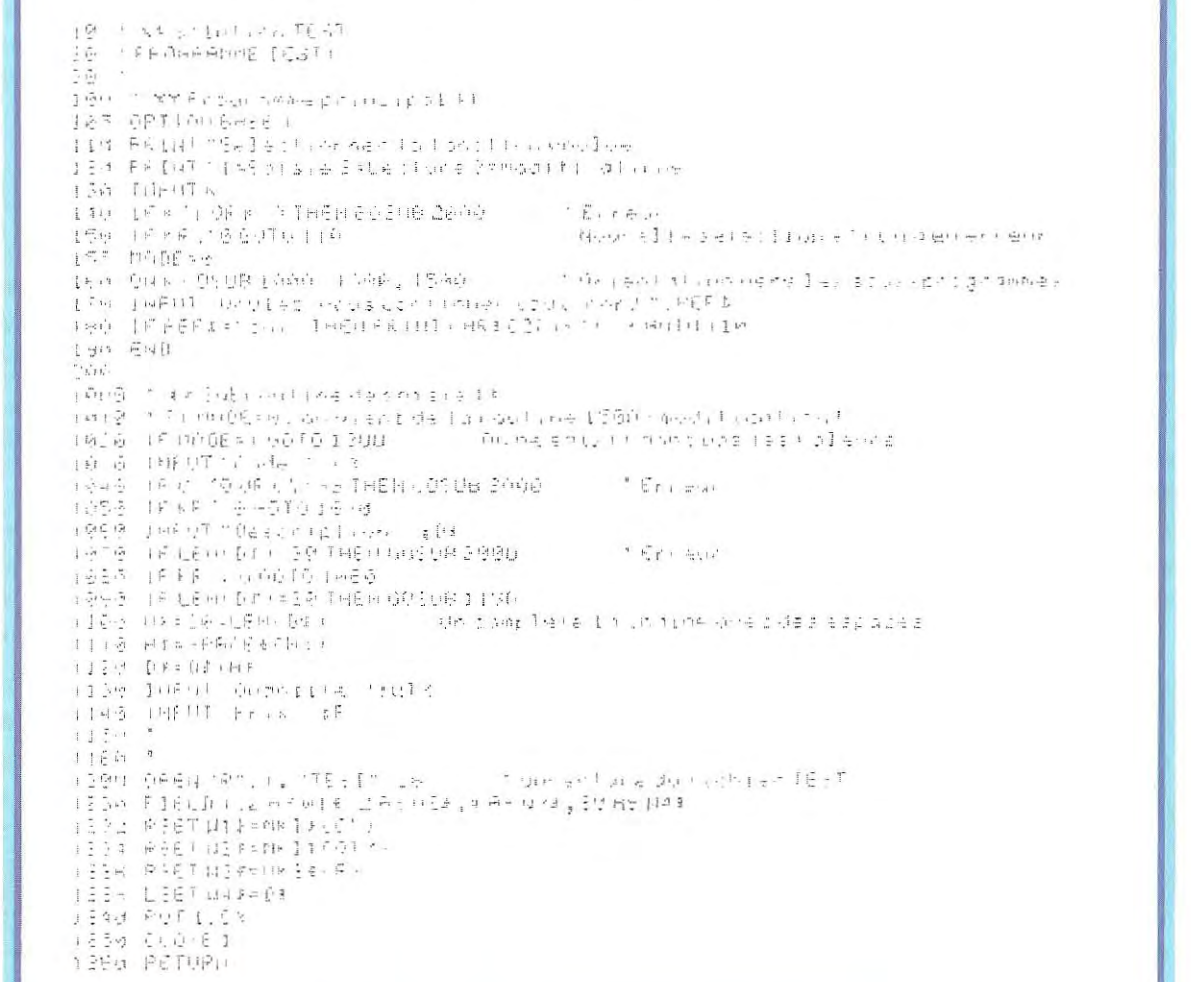
---

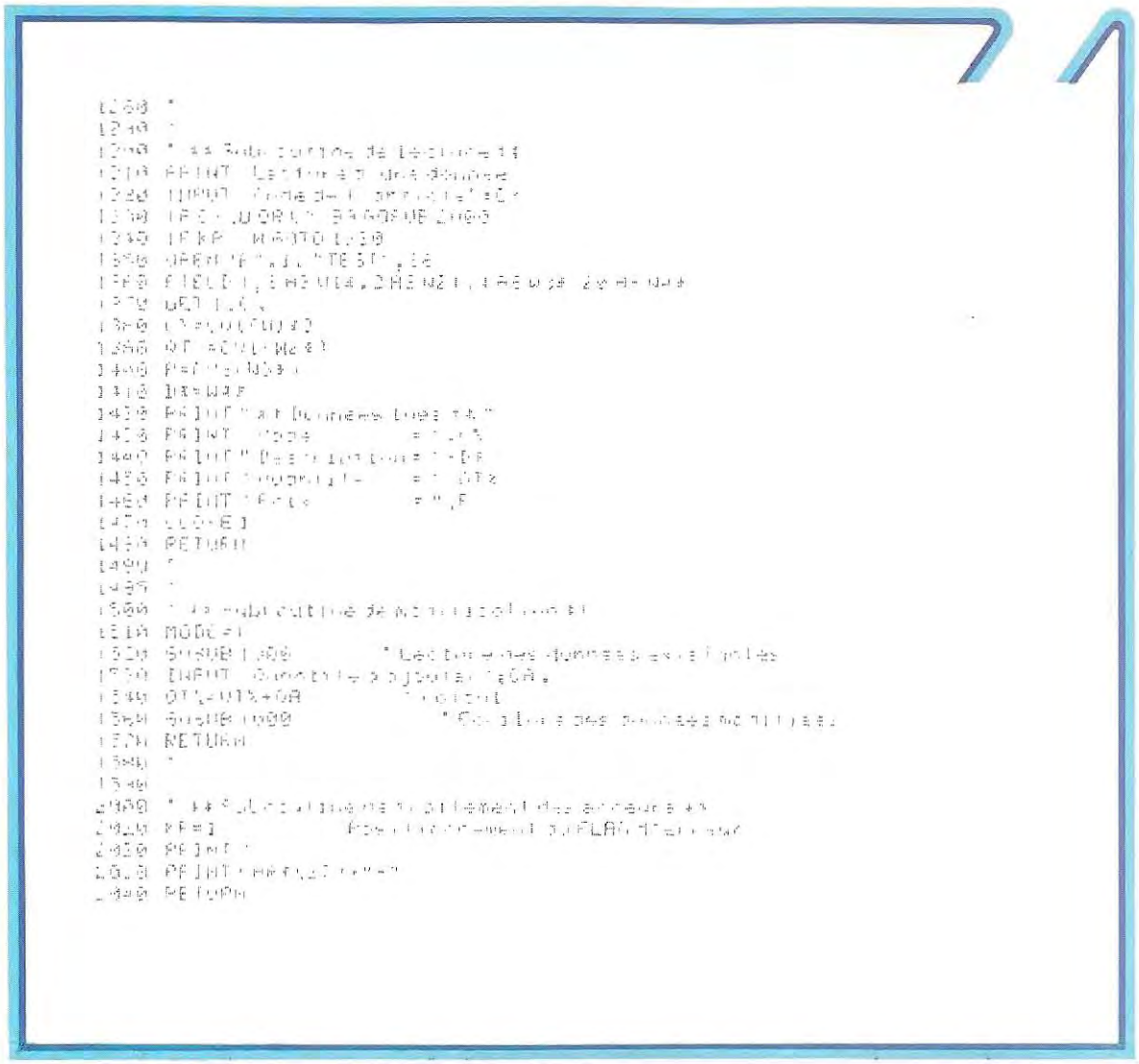
2 / a) génère une erreur système et stoppe l'exécution du programme ;
b) créé le fichier.

---

3 / On doit utiliser la fonction LOC(N).

4-5 / Le listing du programme répondant aux questions 4 et 5 est illustré ci-dessous et page suivante.
Rappelons que cette solution n'est donnée qu'à titre indicatif. L'écriture dans le détail des instructions d'un programme est généralement très subjective. La forme d'un programme, les instructions ou les fonctions employées dépendent beaucoup des habitudes du programmeur.





Recherche (sous-programme 5000)

En général, la complexité des sous-programmes de recherche dépend étroitement du volume des données à traiter. Si le fichier de données est de grande dimension, on aura intérêt à l'ordonner au préalable, pour faciliter ensuite la recherche. S'il s'agit d'un répertoire, par exemple, on classera les noms par ordre alphabétique ; la recherche d'un nom particulier sera ensuite effectuée très rapidement.
Dans les applications sur micro-ordinateur, le volume des données constitue rarement un obstacle ; dans ce cas, il sera plus simple d'examiner le fichier en entier, sans le trier au préalable, et d'extraire les enregistrements répondant aux critères recherchés, au fur et à mesure qu'ils se présentent. Par cette technique, on dépense plus de temps que nécessaire, puisque tous les enregistrements sont lus (même ceux qui auraient été éliminés avec un fichier ordonné), mais la différence en temps d'exécution ne justifie pas toujours de se lancer dans les difficultés d'une programmation de haut niveau. Pour avoir un ordre de grandeur, sachant que le système lit environ une centaine d'enregistrements à la minute, on peut considérer que le gain de temps éventuel est de l'ordre de la minute.
La complexité des sous-programmes de recherche dépend en fait des degrés de liberté nécessaires à l'application, c'est-à-dire du nombre de paramètres que l'on veut utiliser comme clés d'accès aux données.

Par exemple, dans notre fichier d'adresses, nous pouvons choisir comme élément de sélection le nom seul, ou sélectionner également sur la ville. Une telle méthode impliquerait l'écriture d'un sous-programme spécifique pour chaque critère de recherche.
On aura plutôt intérêt à paramétrer le sous-programme de recherche, puis à rendre ceci transparent pour l'utilisateur en affichant la liste des rubriques de chaque enregistrement. L'utilisateur peut alors choisir le critère de sélection, en entrant la rubrique concernée.
Cette méthode a été utilisée dans le sous-programme 5000 (voir organigramme ci-dessous) où toutes les rubriques d'un enregistrement sont passées en revue. L'utilisateur peut alors soit exclure la rubrique des critères de choix (caractère *), soit la valider en entrant la valeur à rechercher pour ce critère.

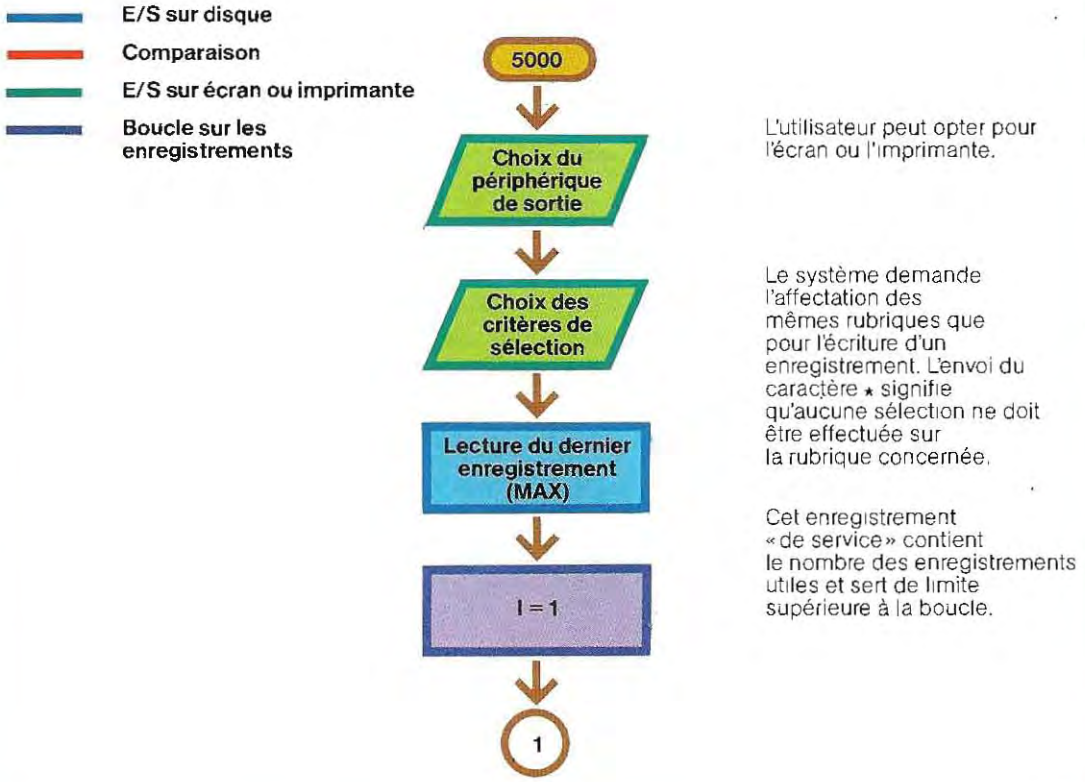
Impression (sous-programme 6000)

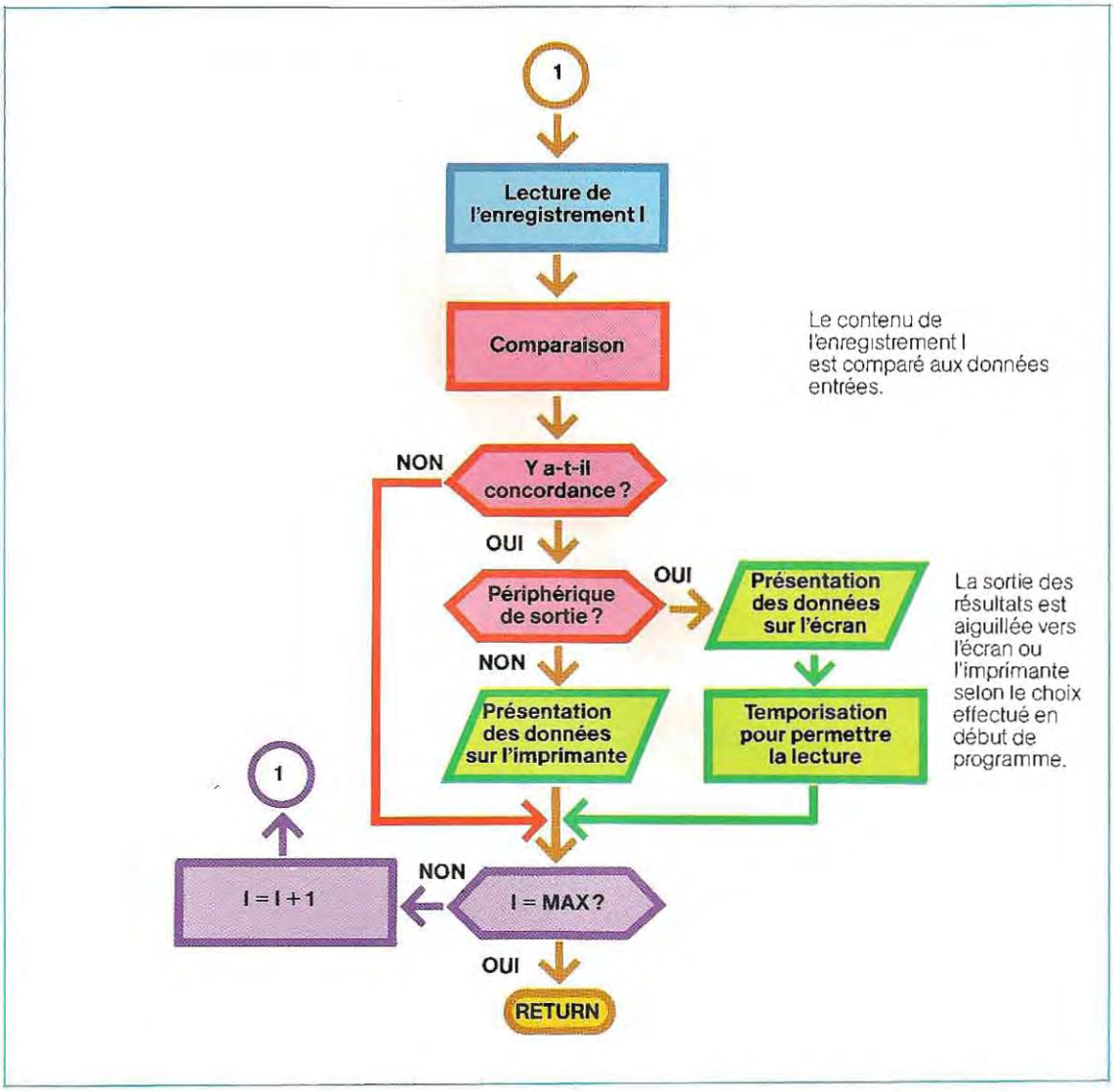
Le sous-programme d'impression doit permettre l'édition sur papier de tous les enregistrements du fichier. Il sera donc constitué d'une boucle allant du premier au dernier enregistrement écrit. Le numéro de ce dernier enregistrement est mémorisé dans une zone supplémentaire du fichier. Cette procédure ne présentant pas de difficultés particulières, nous laissons au lecteur le soin de tracer lui-même l'organigramme selon la structure qui lui semble la mieux adaptée à l'édition. On pourrait également utiliser une routine d'impression paramétrée dont nous présenterons les modalités par la suite.

Opérations d'impression particulières

Le programme que nous venons de décrire est adaptable à de multiples applications et ne nécessite pour cela que des modifications mineures (nombre, signification et longueur des variables). Par contre, la routine d'impression est difficilement aménageable pour certains types d'édition. Les fichiers d'adresses,

SOUS-PROGRAMME DE RECHERCHE





par exemple, sont souvent utilisés pour des expéditions en nombre de courrier ou d'imprimés. Les sorties du programme, c'est-à-dire les noms et adresses, doivent alors être réalisées sur un support directement utilisable.
A cet effet on trouve dans le commerce des bandes de papier sulfurisé, adaptées au système d'entraînement par picots et sur lesquelles sont disposées des rangées d'étiquettes autocollantes. Ce système est beaucoup plus rapide et universel que l'impression d'enveloppes sur support spécifique (enveloppes, bandes de routage, etc.).
Le sous-programme d'impression doit toutefois être modifié en conséquence car on doit tenir compte d'un format très particulier dû à la disposition des étiquettes. Il faut, en effet, structurer les sorties de façon à faire imprimer plusieurs noms sur une même ligne – un sur chaque étiquette, puis les adresses correspondantes sur la ligne suivante et enfin les villes sur une troisième ligne. La routine d'impression doit donc opérer sur des groupes d'enregistrements équivalents au nombre d'étiquettes présentes sur la largeur de la feuille. Par conséquent, il faudra prévoir deux boucles imbriquées : une principale assurant la saisie de tous les enregistrements à partir du disque et une secondaire qui regroupera les données par une ligne d'impression et effectuera les sorties rubrique par rubrique.
Ce principe d'impression des étiquettes est illustré par le schéma de la page 675 pour une rangée de quatre étiquettes. Dans cet

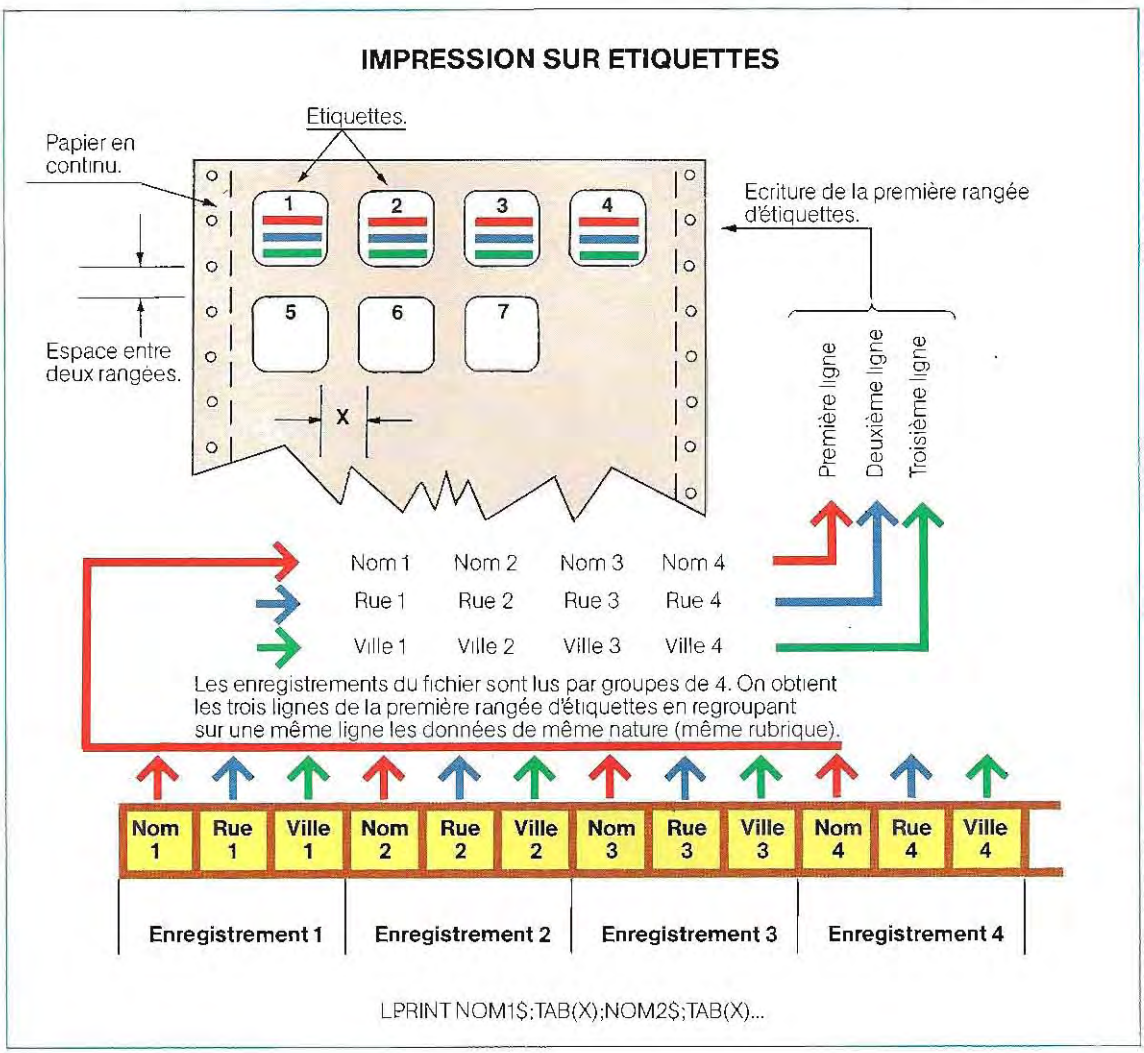
exemple, l'incrément de la bouche principale est 4 et les trois rubriques correspondant aux noms et prénoms, aux rues et aux villes sont traitées par une boucle intérieure. Le module d'impression doit contenir des instructions de tabulation permettant le centrage de l'adresse sur les étiquettes.

L'agencement des données

Pour gérer des fichiers, il est indispensable de disposer de routines spécialisées dans le tri des données à partir d'une ou plusieurs rubriques.

Ces routines d'usage général, appelées programmes de tri, sont souvent fournies systématiquement avec le matériel et sont, dans tous les cas, très faciles à se procurer. Il peut cependant s'avérer utile de disposer de sous-programmes personnels mieux adaptés à un usage particulier, notamment quand les données à traiter sont en nombre limité. De même, il est souvent intéressant de pouvoir ordonner des tables présentes dans la mémoire centrale. Or, les utilitaires de tri travaillent sur des fichiers conservés dans des mémoires de masse et leur emploi en RAM est beaucoup moins performant. Signalons toutefois qu'un algorithme de tri écrit en Basic n'est envisageable qu'avec un Basic compilé. L'interprétation demande en effet beaucoup trop de temps pour ce type d'applications, la seule alternative étant alors le langage Assembleur.

Nous allons maintenant présenter deux programmes de tri. Le premier travaille en mémoire et ne peut traiter un fichier externe que dans la mesure où celui-ci peut être transféré intégralement dans la mémoire. Le



second est par contre conçu pour des fichiers sur disque et utilise une méthode de recherche dichotomique. C'est un sous-programme performant adapté à la gestion de fichiers de taille moyenne.

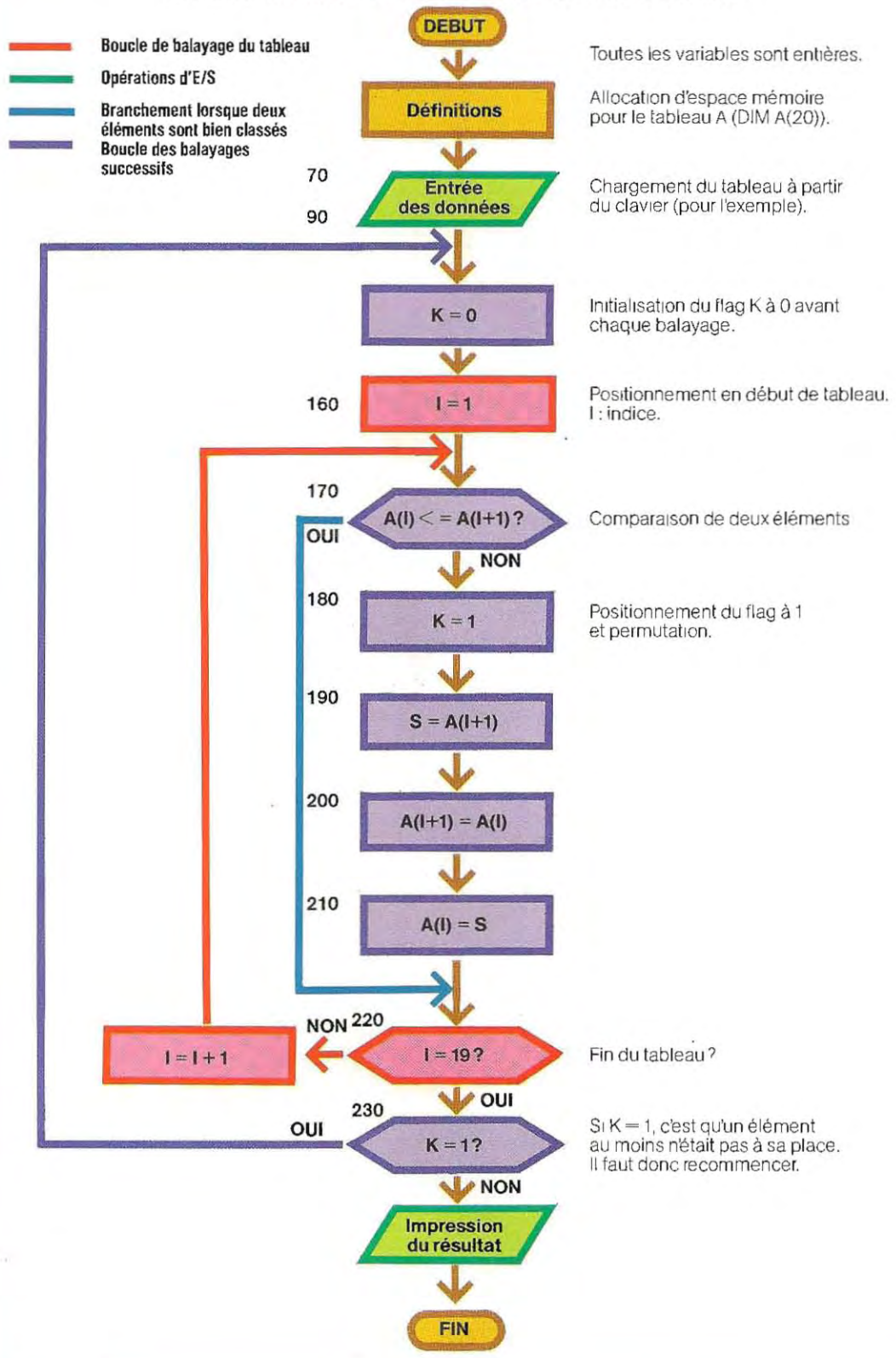
Tri en mémoire

L'organigramme représenté page 677 est celui d'un programme qui classe par ordre croissant le contenu d'un tableau à 20 éléments : A(20).
Le classement s'effectue par comparaisons successives de chaque élément du tableau avec l'élément suivant (d'indice immédiatement supérieur) : si le premier est le plus grand, les deux éléments sont intervertis. Les valeurs les plus importantes du tableau sont ainsi peu à peu acheminés vers la fin de celui-ci. Ensuite un nouveau balayage a lieu, et ainsi de suite jusqu'à ce qu'un passage n'ait donné lieu à aucune inversion. Chaque élément du tableau est alors inférieur au suivant et le tri est terminé. Pratiquement, on utilise un flag K qui est positionné à 1 dès qu'une inversion a lieu. La fin d'un balayage (comparaison des deux derniers éléments) assortie d'un flag K=0 indique donc qu'aucune permutation n'a eu lieu et qu'on peut donc sortir de la boucle pour terminer le programme.
Cette routine, dont le listage se trouve pages 678 et 679, n'a qu'une simple valeur d'illustration. Elle pourrait, en effet, être simplifiée par une utilisation plus rationnelle du flag K.
Cet indicateur ne peut prendre que deux valeurs : 1 et 0, associées à Vrai et Faux puisque son positionnement est lié à une condition (la réalisation d'une permutation au cours d'un passage ou l'achèvement du classement). Une boucle, devant être répétée tant qu'une condition n'est pas remplie, est un cas typique d'application de l'instruction WHILE... WEND.
Vous trouverez donc dans la seconde partie de la page 679 une autre version du programme où l'on a utilisé cette instruction. Une seconde modification a été introduite dans ce programme avec l'emploi de la fonction SWAP. Dans la première version, on avait dû prévoir une mémoire de travail (S) pour permuter le contenu de deux éléments successifs du tableau A(I) et A(I+1). Avec l'instruction SWAP, la gestion de cette mémoire est prise en charge par le système et l'écriture s'en trouve allégée.
La permutation du contenu de deux variables nécessite, en effet, le transfert momentané de

Salle de contrôle d'une importante gare.

K. Reese/Marka

ORGANIGRAMME DE CLASSEMENT D'UN TABLEAU



CLASSEMENT D'UN TABLEAU (PREMIERE VERSION)

```
10 ' ** PROGRAMME CLASS1
20 OPTION BASE 1
30 DIM A(20)
40 ' ** ENTREE DES DONNEES A CLASSER
50 '
60 FOR I=1 TO 20
70 PRINT " Donnee N.: ";I,
80 INPUT "VALEUR     ";A(I)
90 NEXT I
100 '
110 ' ** IMPRESSION DES VALEURS ENTREES
120 '
130 LPRINT " TABLEAU ENTRE "
140 LPRINT
150 FOR I=1 TO 20
160 LPRINT " I = 1 ";I," Valeur = ";A(I)
170 NEXT I
180 '
190 ' *** CLASSEMENT
200 '
210 K=0
220 FOR I=1 TO 19
230 IF A(I)<=A(I+1) GOTO 270
240 K=1
250 S=A(I+1)
255 A(I+1)=A(I)
260 A(I)=S
270 NEXT I
280 IF K=1 GOTO 210
290 '
300 ' ** IMPRESSION DES DONNEES CLASSEES
310 LPRINT " TABLEAU CLASSE "
320 LPRINT
330 FOR I=1 TO 20
340 LPRINT " I = ";I," Valeur = ";A(I)
350 NEXT I
360 END
```

```
 TABLEAU ENTRE

I = 1           Valeur = 32
I = 2           Valeur = 25
I = 3           Valeur = 18
I = 4           Valeur = 0
I = 5           Valeur = 5
I = 6           Valeur = 48
I = 7           Valeur = 98
I = 8           Valeur = 74
I = 9           Valeur = 125
I = 10          Valeur = 2
I = 11          Valeur = 56
I = 12          Valeur = 36
I = 13          Valeur = 14
I = 14          Valeur = 25
I = 15          Valeur = 12
I = 16          Valeur = 10
I = 17          Valeur = 91
I = 18          Valeur = 58
I = 19          Valeur = 37
I = 20          Valeur = 51
```

```
TABLEAU CLASSE

I = 1          Valeur = 0
I = 2          Valeur = 2
I = 3          Valeur = 5
I = 4          Valeur = 10
I = 5          Valeur = 12
I = 6          Valeur = 14
I = 7          Valeur = 18
I = 8          Valeur = 25
I = 9          Valeur = 25
I = 10         Valeur = 32
I = 11         Valeur = 36
I = 12         Valeur = 48
I = 13         Valeur = 56
I = 14         Valeur = 58
I = 15         Valeur = 61
I = 16         Valeur = 74
I = 17         Valeur = 77
I = 18         Valeur = 91
I = 19         Valeur = 98
I = 20         Valeur = 125
```

CLASSEMENT D'UN TABLEAU (SECONDE VERSION)

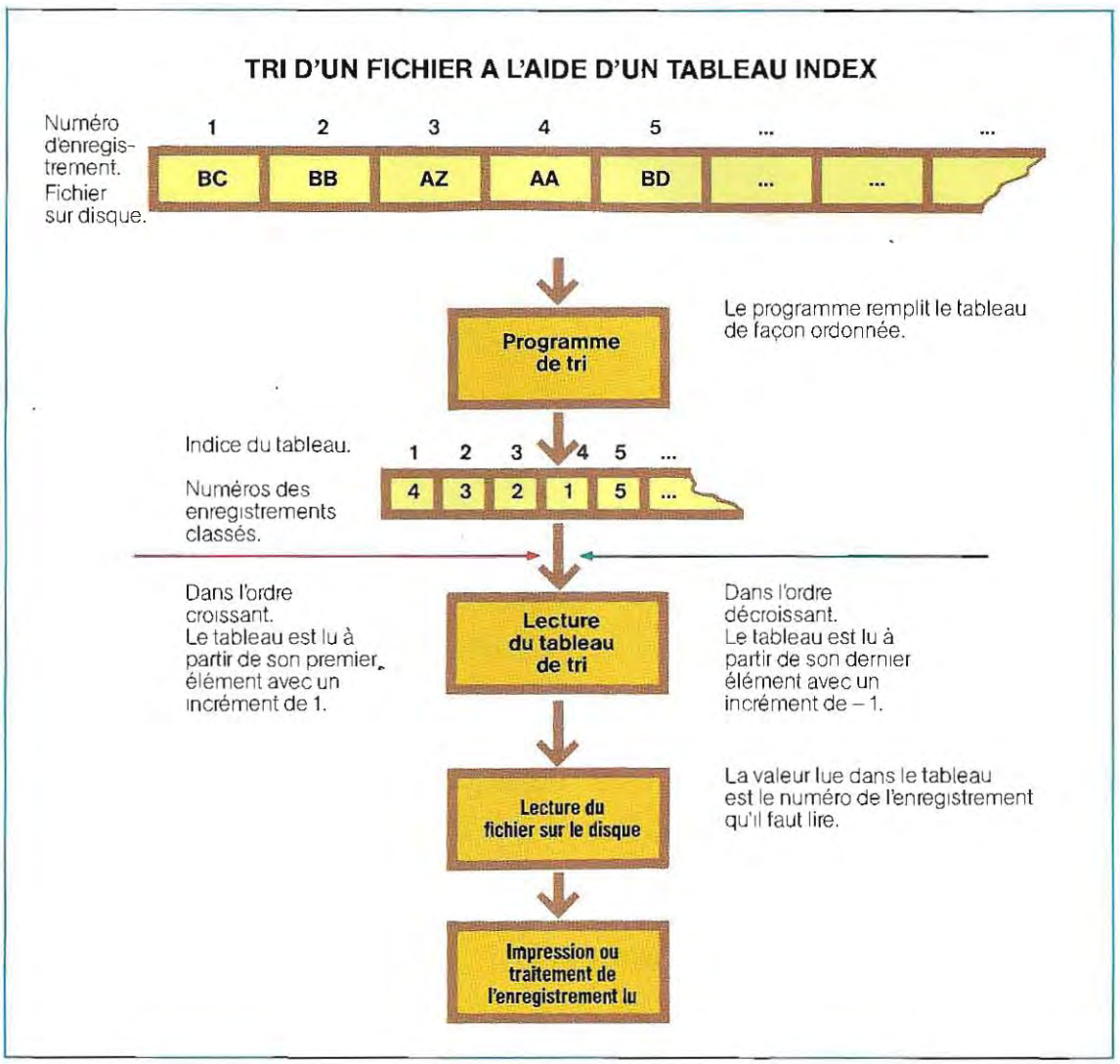
```
10 ' ** PROGRAMME CLA[illegible]
20 [illegible]
30 DIM A(20)
40 ' ** ENTREE DES DONNEES A CLASSER
50
60 FOR I=1 TO 20
70 PRINT " Donnee N. :";I,
80 INPUT "VALEUR      ";A(I)
90 NEXT I
100 '
110 ' ** IMPRESSION DES VALEURS ENTREES
120 '
130 LPRINT " TABLEAU ENTRE "
140 LPRINT
150 FOR I=1 TO 20
160 LPRINT " I = ";I," Valeur = ";A(I)
170 NEXT I
180 '
190 ' *** CLASSEMENT
200 '
210 K=1 ' On place K a 1 pour que la boucle s'execute au moins une fois
220 WHILE K
230 K=0
240 FOR I=1 TO 19
250 IF A(I)>A(I+1) THEN SWAP A(I),A(I+1):K=1
260 NEXT I
270 WEND
290 '
300 ' ** IMPRESSION DES DONNEES CLASSEES
310 LPRINT " TABLEAU CLASSE "
320 LPRINT
330 FOR I=1 TO 20
340 LPRINT " I = ";I," Valeur = ";A(I)
350 NEXT I
360 END
```

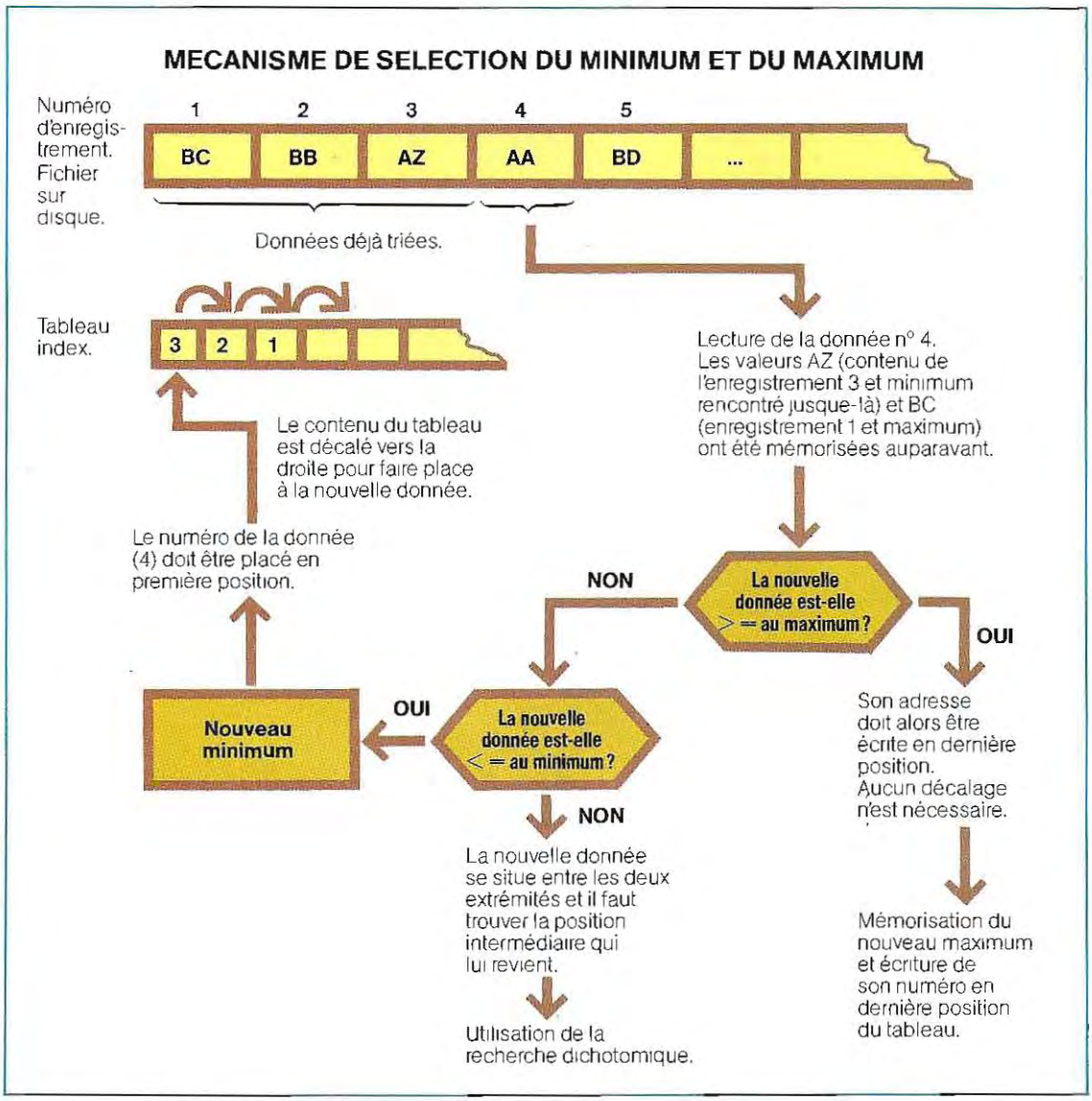
l'une des valeurs à échanger dans une troisième variable, dite mémoire de travail. L'instruction SWAP permet d'obtenir le même résultat par simple interversion du nom des deux variables. La zone mémoire de départ prend ainsi le nom de la zone d'arrivée et inversement. Le but initial – faire passer les données d'une variable à une autre – est donc également atteint, mais inverser les adresses de deux variables est beaucoup plus rapide que d'en réaffecter trois.

On peut adapter ce programme à tout tableau, quel que soit le nombre de ses éléments – dans les limites de l'espace mémoire disponible. Il suffit pour cela de remplacer les nombres 20 et 19 (lignes 30, 150, 240 et 330) par les valeurs appropriées ou un paramètre défini au préalable. Enfin, le programme tel qu'il est rédigé ne s'applique qu'à des valeurs numériques mais son adaptation à des chaînes de caractères est aisément réalisable en définissant comme telles (symbole $) la variable indicée A(N) et la mémoire de travail S. Rappelons en effet que les symboles > et < employés sur des chaînes de caractères opèrent selon l'ordre alphabétique.

Tri d'un fichier sur disque

Dans le programme qui va être présenté, on utilise comme mémoire de travail un tableau qui doit être dimensionné de façon à contenir autant d'éléments que le fichier comporte d'articles. En effet, chaque fois qu'un enregistrement sera lu, son numéro sera transféré dans ce tableau à la place qui lui revient en fonction du contenu alphabétique de cet





enregistrement. Donc, si, par exemple, le contenu de l'enregistrement n° 15 est inférieur (selon l'ordre alphabétique) à ceux du n° 1 et du n° 2, la valeur 15 sera positionnée dans le tableau de tri avant les valeurs 1 et 2.

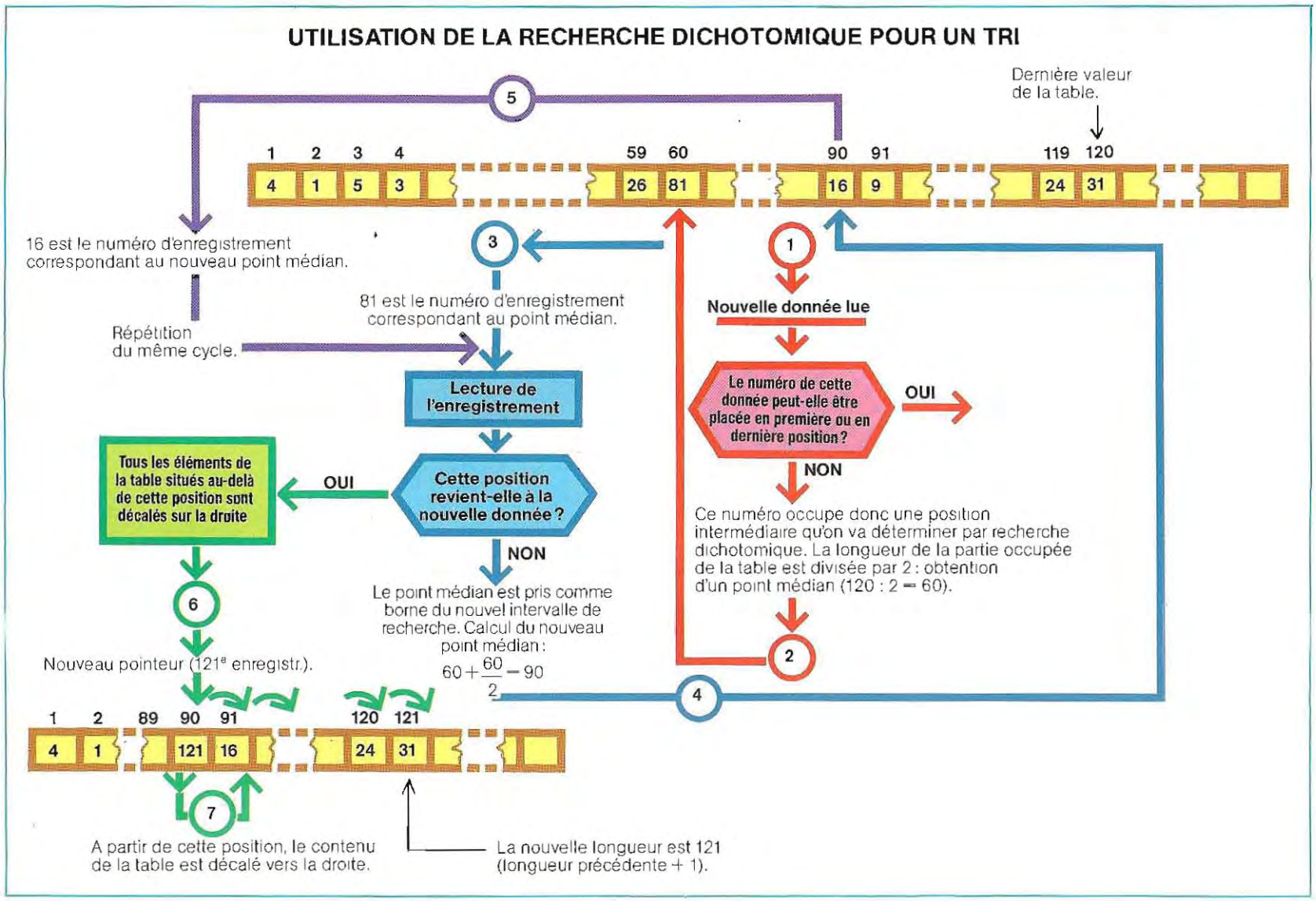
On peut ensuite utiliser les données de ce tableau comme des pointeurs. Selon qu'on remontera ou qu'on descendra le tableau, on aura accès aux enregistrements classés en ordre alphabétique croissant ou décroissant. On dit que ce tableau constitue un index. La logique du traitement est illustrée ci-dessous. Le remplissage du tableau de tri se fait de la façon suivante.

Supposons que plusieurs enregistrements aient déjà été traités, les trois premiers par exemple. Le tableau contient dorénavant trois valeurs, qui sont les numéros de ces enregistrements classés dans l'ordre croissant de leur contenu. Le premier élément du tableau contient donc le numéro de la plus petite donnée déjà triée, le second élément celui de la donnée immédiatement supérieure et ainsi de suite.

On extrait ensuite la donnée suivante (enregistrement numéro 4). On peut, bien sûr, déterminer sa place dans le tableau en relisant tous les enregistrements déjà traités et en comparant leur contenu à cette nouvelle donnée.

Cependant, il existe deux méthodes dont l'utilisation conjointe accélère la recherche. La

UTILISATION DE LA RECHERCHE DICHOTOMIQUE POUR UN TRI



première consiste à mémoriser la plus grande et la plus petite des données déjà triées et à comparer systématiquement chaque nouvelle donnée à ces deux extrêmes (voir schéma page 681).
Le numéro d'un enregistrement inférieur ou égal au minimum sera placé en première position et celui d'un enregistrement supérieur ou égal au maximum en dernière position.
Si ces deux comparaisons n'ont rien donné, c'est que l'enregistrement en question correspond à une position intermédiaire.
On emploiera alors la seconde méthode dont nous avons parlé : la recherche dichotomique (divisions successives par deux). Après avoir extrait la valeur médiane du tableau, on lit l'enregistrement du fichier correspondant à ce numéro (BB) et on le compare à la nouvelle donnée. On décide ensuite de continuer la recherche dans la partie droite ou gauche du tableau, selon le résultat de la comparaison.
Un schéma général du programme est représenté page 682.
Des possibilités complémentaires sont généralement offertes dans les logiciels de tri proposés dans le commerce. La description du logiciel utilitaire OLISORT* élaboré par Olivetti donnera une idée des performances de tels outils.

Le programme utilitaire OLISORT

Conçu pour l'ordinateur individuel Olivetti M20, Olisort est un progiciel de tri/fusion performant et d'une grande souplesse.
Une vaste gamme de fonctions le rend apte à la plupart des applications de gestion. Voici quelques-unes des possibilités d'Olisort :

- tri d'un fichier ;
- fusion de deux fichiers ;
- sélection d'un enregistrement dans un fichier ;
- concaténation d'un fichier à la fin d'un autre ;
- réalisation simultanée d'un tri et d'une sélection.

Les commandes d'Olisort sont constituées de paramètres qui sont transférés dans une chaîne de caractères. Cette chaîne peut être communiquée au logiciel par le biais d'instructions Basic ou par une séquence de

* OLISORT est une marque déposée.

Le système de calcul MOT 53385 sur IBM.

Grazia Neri/Photo Media

Communiquer à l'aide d'un micro-ordinateur

Remarquable outil de communication, le micro-ordinateur peut servir à pallier les handicaps des personnes qui souffrent de troubles moteurs, ou de troubles de la compréhension ou de la phonation, comme le montrent certaines expériences menées tant en Europe qu'aux Etats-Unis et au Japon. Ces programmes sont l'aboutissement de travaux poursuivis en collaboration entre les spécialistes de la rééducation fonctionnelle et les chercheurs en électronique.

Aide à l'expression ou à la compréhension, la machine est dans tous les cas un précieux moyen pour permettre aux handicapés de communiquer avec leur entourage et d'évoluer dans leur environnement.

Pour un aveugle, ce sera par exemple l'intérêt de l'ascenseur (ISI) ou du feu rouge (SILEC) parlants, des jeux électroniques parlants (ISI), de la calculatrice dotée de la parole (ISI, Kurzweil).

Il s'agit là de ***synthèse acoustique,*** *tandis que, grâce à la* ***synthèse phonétique,*** *on réalise des machines à écrire parlantes qui disent le mot qui vient d'être tapé (IBM-France), autorisant les aveugles à les utiliser normalement pour tous les travaux spécifiques du traitement de texte.*

Dans le cas des infirmes moteurs cérébraux, la synthèse joue le rôle d'une ***prothèse vocale.*** *C'est un moyen très efficace pour l'apprentissage de la lecture et de l'écriture.*

Selon les pays et selon le matériel utilisé, le système peut fonctionner différemment mais il comporte toujours une base fondamentale assez similaire et nous nous bornerons à décire quelques modèles ou programmes français et italiens.

En ce qui concerne la France, citons le système à clavier complet sur lequel on forme les mots (Handivoice-Votrax, Sahara du CNET qui fonctionne avec un langage Bliss) à l'aide de touches correspondant à des lettres, des mots ou des symboles, et le système à touches de sélection (en nombre très réduit : deux en général ou même une seule), qui offre une manipulation très aisée, particulièrement adapté aux besoins des grands handicapés moteurs (Vidéocom du professeur Eschstruth, et fonctionnant avec le synthétiseur Icolog développé par le LIMSI-CNRS).

Le Vidéocom permet la sortie d'un texte par affichage sur un écran vidéo ou sur imprimante. Les entrées peuvent se faire de deux manières différentes. Pour l'utilisateur dont la motricité n'est pas trop gravement atteinte, l'entrée se fait par clavier, sur lequel le texte est composé assez rapidement. Mais si l'utilisateur souffre d'une motricité très perturbée ou très limitée, la commande par une ou deux touches permet de procéder à une sélection progressive parmi les soixante lettres ou symboles disponibles, en partant du choix d'une des douze colonnes possibles. Le système, bien que moins rapide, est une première amélioration des conditions de la communication des grands handicapés. Grâce au synthétiseur Icolog qui accepte directement le texte en français, l'utilisation peut avoir recours à une orthographe simplifiée, ce qui lui fera gagner du temps. Il peut écrire, par exemple : « je veu alé ché moi ».

Le système italien Logos 4, développé par l'université de Gênes, est assez proche dans son principe.

Des études statistiques faites sur le langage, et particulièrement sur les textes destinés aux enfants de l'école primaire ou écrits par eux, ont permis d'établir des listes de mots et de groupes de lettres rassemblées selon leur fréquence dans le langage. A partir d'un premier choix fait par l'utilisateur, ce dernier peut visualiser à l'écran les lettres qui pourraient venir ensuite en ordre de probabilité décroissante. Il dispose alors, pour choisir le caractère adéquat, soit de deux touches – une touche de rejet et une touche d'acceptation –, soit d'une seule touche de sélection (dans ce cas, les lettres proposées par la machine se succèdent à un certain rythme, modifiable). La séquence de lettres affichée varie sans cesse puisque l'étude statistique du vocabulaire permet de déterminer quelle lettre a la plus grande probabilité de venir compléter un mot. On évite ainsi de parcourir inutilement l'alphabet et on parvient à communiquer assez rapidement dès qu'on a acquis une certaine habitude.

Deux autres touches sont utilisées : la première pour inverser l'ordre des caractères et la seconde pour passer du balayage automatique au balayage manuel. Les lettres sont alors affichées en différentes couleurs. Leur taille

diminue au fur et à mesure qu'elles se rapprochent (graphisme « en entonnoir »). L'utilisateur peut ainsi préparer son choix, en s'aidant de ce qu'il voit sur l'écran. En outre, il est possible d'ajouter à cette représentation une amplification sonore et de ralentir la vitesse de balayage pour laisser aux infirmes moteurs les plus handicapés le temps de transmettre leur choix.

La différenciation des lettres par une couleur présente un intérêt pour les utilisateurs ayant également des problèmes de vision.

Les lettres déjà choisies pour un même mot sont visualisées en couleur, à gauche de l'écran. Il est ainsi plus facile de terminer le mot et éventuellement de le corriger. Une fois composé, le mot apparaît à la suite des lignes précédentes du texte, qui sont affichées au bas de l'écran en plus petits caractères.

En plus des vingt-six lettres de l'alphabet, plusieurs symboles ont été ajoutés pour désigner les fonctions d'espacement, d'alinéa, de retour en arrière, de correction, d'impression du texte (qui vient d'être écrit et qui n'est pas mémorisé par la machine) et de retour aux fonctions initiales du programme.

Un programme comme le Logos 4 (mis au point pour un matériel IBM) ne peut être appliqué que si la configuration de l'ordinateur utilisé comporte au moins les éléments suivants :

Une unité centrale disposant de 128 K de RAM et d'un connecteur d'entrée analogique (manettes de jeux) ;

un lecteur de disquettes 180 K ;

un écran grahique ;

un clavier simplifié à larges touches pour handicapés ;

un clavier ;

une imprimante.

L'écran utilisé peut être celui d'un poste de télévision connecté au système par un modulateur.

On pourra augmenter la taille de la configuration minimale, en lui ajoutant une seconde unité de disque de 180 K ou bien en remplaçant la première par une unité de 360 K.

Par ailleurs, il faut prévoir des claviers spéciaux différents (très grands ou pneumatiques, par exemple) afin de les adapter à chaque cas particulier.

Chaque utilisateur a la possibilité d'obtenir une disquette programmée en fonction de ses propres capacités motrices. Il existe, en effet, deux masques de saisie qui permettent d'opter pour telle ou telle caractéristique d'affichage ou d'impression.

L'un de ces marques offre le choix entre différentes modalités de contrôle du mouvement et de présentation des caractères.

Les lettres peuvent se présenter de deux façons : en balayage automatique, elles se déplacent automatiquement de droite à gauche à une vitesse programmable (un déplacement par seconde, et jusqu'à un toutes les cinq secondes), alors qu'en balayage manuel, l'utilisateur doit exprimer son choix pour chaque lettre – acceptation ou refus – en appuyant sur la touche « oui » ou sur la touche « non » : il doit donc être en mesure d'actionner deux touches.

En revanche, avec le balayage automatique, la sélection des lettres ne demande à l'utilisateur qu'un effort minimal (un seul mouvement suffit généralement).

Pourtant, le balayage manuel permet d'aller plus vite et de commander directement la machine. Il est possible de faciliter la perception des informations visuelles en modifiant la couleur du fond (seize possibilités), des lettres de l'« entonnoir » (quatre couleurs), de la lettre en cours de sélection (quatre couleurs) et des lettres déjà acquises (quatre couleurs). On peut ainsi adapter la présentation aux facultés visuelles de l'utilisateur.

En outre, on peut faire varier la combinaison de couleurs pour augmenter ou diminuer la difficulté, en fonction des capacités de perception et d'apprentissage de l'utilisateur.

Enfin, on peut faire disparaître les lettres de l'entonnoir ou les lettres déjà choisies en les écrivant de la même couleur que le fond.

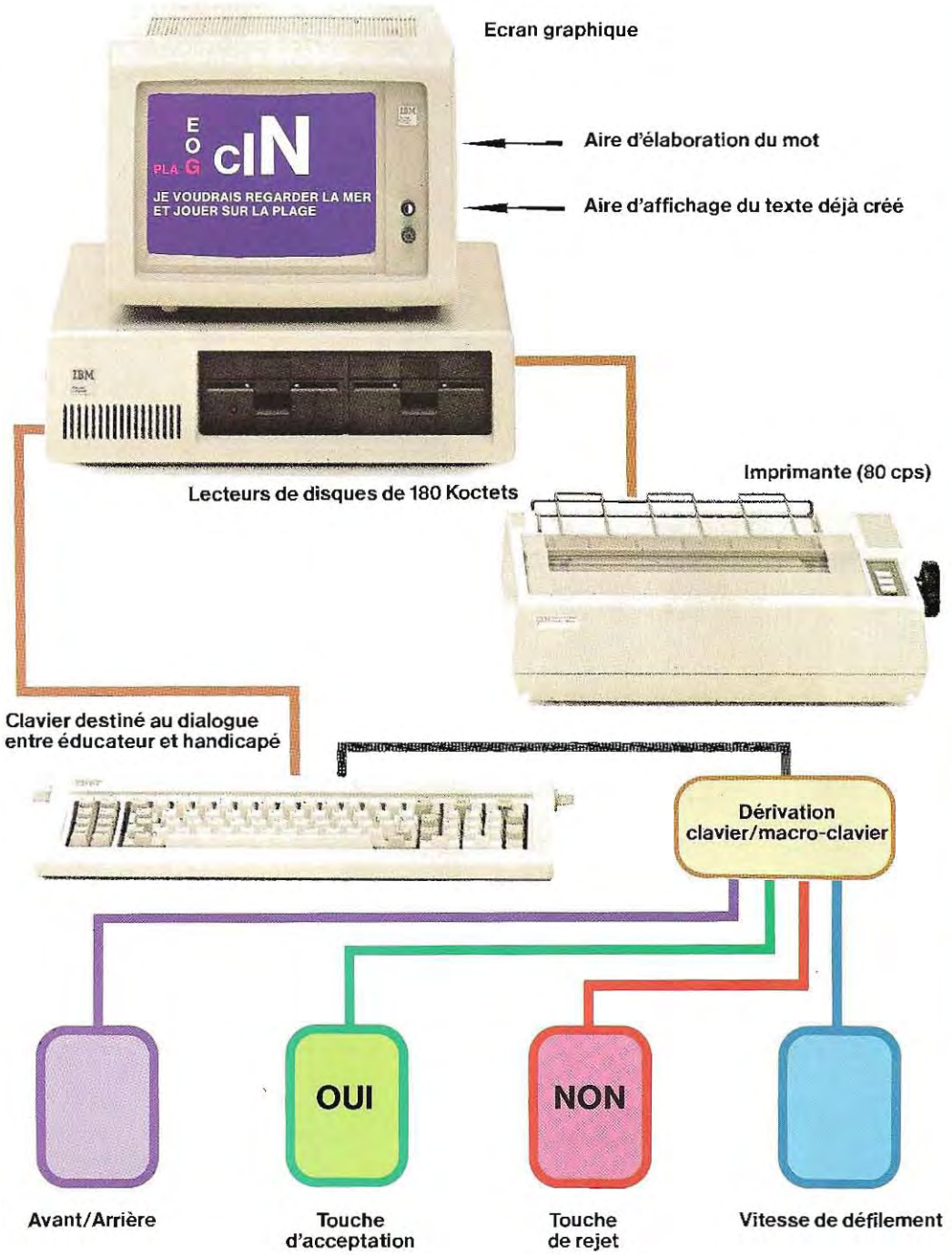
Quant à l'impression, on peut programmer sur l'imprimante la taille des caractères et des interlignes, ainsi que l'intensité du trait.

Au début de l'exécution du programme, le choix est donné à l'utilisateur entre sept fonctions qui permettent une gestion autonome du programme lui-même.

On peut commencer un nouveau texte, le mémoriser éventuellement, reprendre un travail interrompu, rappeler un texte sauvegardé sur disque ou imprimer la totalité d'un texte.

La première fonction de communication est la fonction « d'appel ». Elle commande un signal acoustique intermittent qui avertit l'édu-

CONFIGURATION DU MATERIEL NECESSAIRE AUX PROGRAMMES D'AIDE PAR L'ORDINATEUR A LA COMMUNICATION DES HANDICAPES



Macro-clavier ou autre périphérique d'entrée adapté aux aptitudes de l'handicapé.

IBM

L'élaboration d'un message à l'aide du programme Logos 4. Les espoirs fondés sur le micro-ordinateur pour la rééducation des handicapés se concrétisent aujourd'hui. Les expériences menées tant en Europe qu'aux Etats-Unis ou au Japon, révèlent d'immenses possibilités pour abattre le mur d'incommunicabilité qui isole les handicapés.

cateur quand le sujet demande à être aidé. De plus, éducateur et handicapé peuvent dialoguer, le premier à l'aide de son clavier et le second au moyen de ses touches spéciales. L'échange de questions et de réponses affichées à l'écran aide beaucoup l'élève à se familiariser avec le programme.

Il est clair, par ailleurs, que le travail sur l'ordinateur apprend progressivement à l'handicapé à se réinsérer dans son milieu de vie, par la communication avec les personnes mais aussi par sa possibilité de commander à son environnement.

Des systèmes comme Mozart du LIMSI-CNRS, Seraphine du CNET, pour la France ; ou Logica en Angleterre ; Nec au Japon ; ou encore Verbex aux Etats-Unis, permettent à ces handicapés (tétraplégiques par exemple), de commander l'ouverture des portes, le déplacement d'un lit, la marche d'une voiture, mais aussi les mouvements de leurs propres prothèses, sur ordre oral. L'individu est alors en situation de reconquérir une autonomie proche de la normale et peut envisager d'apprendre un métier qu'il pourra pratiquer. Il ne faut pas s'étonner que les métiers de l'informatique figurent en bonne place dans cette perspective.

Les systèmes dont nous venons de parler sont fondés sur le principe de la ***reconnaissance de la parole*** *par la machine, et certaines expériences sont appliquées au service des malentendants. La machine reconnaît la voix de l'interlocuteur du malentendant et l'affiche sur son écran, où le malentendant la lit, ce qui lui permet un dialogue normal.*

De même, un tel appareil, relié à un téléviseur, permettrait le sous-titrage simultané à l'écran de ce que disent speaker ou personnages du film (étude LIMSI-CNRS en cours).

La machine sert donc d'intermédiaire dans la transmission des messages par affichage ou par impression, mais elle peut aussi fournir de véritables exercices de rééducation vocale, remplaçant le professeur traditionnel.

Elle servira pour ceux qui entendent mais présentent des troubles particuliers relevant de l'orthophonie (retard de la parole chez l'enfant, dyslexie, dysorthographie, ou encore laryngectomie, entre autres). Le micro-ordinateur sert alors d'aide aux enfants sans langage écrit ou sans langage oral. En favorisant la communication autrement que par la parole avec un adulte, il contribuera à débloquer certains obstacles (c'est le cas pour les bègues) et constituera une rééducation efficace.

Dans ce domaine, des expériences très positives ont été tentées en France, par exemple avec Sparte, objet de type automate qui a toujours du succès auprès des enfants.

Il est donc tout à fait regrettable que les grands producteurs en informatique (matériel et logiciel) ne manifestent pas un très grand intérêt dans ce domaine.

LE PROGRAMME OLISORT

Les fonctions du logiciel Olisort sont accessibles soit à partir d'instructions regroupées dans un programme en Basic, soit directement par appel de l'utilitaire Olisortx. Dans ce dernier cas apparaît à l'écran le texte reproduit sur cette page.


```
OLISORTX 1.1 - OLISORT UTILITY PROGRAM
Copyright (c) 1982  OLIVETTI


SELECT FUNCTION
---------------
 S = Sort Existing Data File
 B = Build and Display Command String
 C = Create Command String File
 E = END

Enter desired selection : B
```

Les deux lignes affichées en haut de l'écran informent l'utilisateur du chargement d'Olisort et, plus précisément, de l'utilitaire Olisortx.
L'exécution commence par l'affichage du menu, c'est-à-dire des fonctions disponibles sur cet utilitaire et sélectionnables par la frappe d'une touche

Tri d'un fichier préexistant : touche S.

Définition d'une chaîne constituée de commandes : touche B.

Création d'un fichier regroupant des chaînes commandes : touche C.

Sortie de l'Olisortx : touche E.

A la suite de cette liste, l'opérateur doit frapper la touche correspondant à la fonction choisie. Sur la photo, c'est la touche B qui a été enfoncée. On va donc pouvoir définir une chaîne composée des paramètres et des valeurs conditionnant le tri de données

codes plus succints affectés à des commandes particulières dans un fichier paramètres. Les utilitaires permettent de définir en mode interactif aussi bien le fichier paramètres que les chaînes de commande.
Olisort peut être appelé depuis un programme en Basic ou par l'intermédiaire de l'utilitaire Olisortx en mode direct. On pourra donc, soit créer un programme qui réalise le tri d'un fichier – voire de plusieurs par adjonction de sous-programmes spécifiques –, soit procéder « manuellement » au classement d'un fichier déterminé.
Voici les différentes modalités de tri-sélection autorisées par l'Olisort :

- spécification des critères de tri dans la limite de 10 par enregistrement ;
- sélection ou rejet d'un enregistrement grâce aux clés Select/Exclude (4 au maximum) ;
- deux types d'entrée pour chaque clé : le caractère * qui valide tous les caractères suivants en temps que clé de sélection ou rejet, et le caractère ? qui ne prend en compte que le premier caractère suivant ;
- indication pour chaque clé Select/Exclude de l'opérateur de comparaison (inférieur, égal ou supérieur) ;
- combinaison des différentes clés au moyen des opérateurs logiques OR et AND ;
- possibilité, lors de la comparaison d'une donnée à une clé, d'attribuer la même valeur aux minuscules et aux majuscules ;
- le tri ou la fusion ne débutent pas obligatoirement en début de fichier (on peut sauter jusqu'à 32.767 enregistrements) ;
- disque de travail et disque de sortie peuvent être intervertis sans risque, le programme le signalant avant toute possibilité d'erreur ;
- un traitement des erreurs est programmable car le logiciel mémorise systématiquement tout code d'erreur ;
- l'appel d'Olisort à partir d'un programme Basic est très simple et ne réclame aucune connaissance particulière.

Ajoutons toutefois que ce logiciel est spécifiquement conçu pour l'interpréteur Basic du Olivetti M20 et qu'il n'opère que sur des enregistrements de longueur donnée.

Les prestations. Dans la mesure où l'on ne peut pas effectuer un traitement multivolume, la taille maximale des fichiers auxquels on peut appliquer Olisort est fonction de la capacité du disque. En effet, en cours de tri, le fichier du disque de travail peut atteindre 1,7 fois la dimension du fichier traité. C'est le facteur limitatif. La longueur des enregistrements ne doit pas excéder 256 caractères. Une commande du système PCOS permet toutefois de modifier ce maximum si l'on doit traiter des enregistrements d'une longueur supérieure. Dans les exemples qui suivent, on a systématiquement adopté cette longueur qui est la valeur prise par défaut.
Les clés de tri et les clés Select/Exclude peuvent appartenir aux types de données suivantes :

- chaînes de caractères ;
- valeurs hexadécimales ;
- nombres entiers ;
- nombres en simple précision ;
- nombres en double précision.

Nous avons vu qu'avec une clé alphanumérique, il était possible de supprimer la différenciation entre minuscules et majuscules. Notons que cela n'est pas valable avec une clé de type hexadécimal. La comparaison aura alors lieu sur les codes ASCII des caractères qui sont différents pour les majuscules et les minuscules.
Le tri se fait en fonction des critères définis et en ordre croissant ou décroissant selon la valeur affectée au paramètre correspondant (A pour « ascendant »/D pour « descendant »).
Voici l'exemple du classement d'un fichier clients dans l'ordre croissant des départements (clé majeure) et dans l'ordre décroissant des soldes à régler (clé mineure). L'édition après traitement donne le résultat suivant :

Liste des soldes à régler

Département	Client	Solde (en F)
Ain	M & M Sarl	2596,00
	Boquet Frères	1976,55
	ABC (SA)	568,88
Aisne	NDT (Ets)	8870,77
	Zig et Puce	8345,98
	Pallas (SA)	5689,00
	SOPAC (Sté Nlle)	67,43

Le fichier trié peut être mémorisé sur un autre disque ou, éventuellement, sur le même à la place du fichier initial. Cependant, on risque alors de perdre une partie du fichier, dans le cas d'une défaillance matérielle au cours de l'opération. Il est donc conseillé, par mesure de sécurité, de réaliser cette opération sur une copie de la disquette et de préserver l'original.
Olisort permet également la sélection de tous les enregistrements du fichier qui répondent à certains critères. Cette sélection peut éventuellement être effectuée à l'occasion d'un tri. Nous pourrions, dans notre exemple du fichier clients ordonné, reprendre le même travail, mais en ne retenant que les clients dont le solde est supérieur à 1.000 F. La partie correspondante du fichier obtenu serait alors :

Liste des soldes des clients après sélection

Département	Client	Solde (en F)
Ain	M & M Sarl	2596,00
	Boquet Frères	1976,55
Aisne	NDT (Ets)	8870,77
	Zig et Puce	8345,98
	Pallas (SA)	5689,00

Le nombre maximal de critères de sélection/exclusion employés simultanément est de 4. Ces critères sont totalement indépendants de ceux de tri qui peuvent être effectués en même temps. La sélection peut donc porter sur des rubriques différentes de celles conditionnant le classement. Si seules sont définies des clés de sélection/exclusion, le fichier obtenu en sortie n'aura fait l'objet d'aucun classement et suivra donc l'ordre du fichier d'origine. Quand une chaîne de caractères est employée comme clé de sélection, le programme compare d'abord sa longueur à celle de la rubrique correspondante avant de confronter les contenus des deux chaînes.
Olisort procède de la façon suivante :

1 / il compare les longueurs de la clé et de la zone afin de déterminer, éventuellement, laquelle des deux est la plus longue ;
2 / il tronque alors celle-ci pour obtenir deux chaînes de taille égale ;
3 / enfin, ces deux chaînes sont comparées, caractère par caractère.

Ainsi, avec des enregistrements contenant une rubrique de la forme suivante :

Tuyau de plomb 12 mm × 3 m

On pourra utiliser « Tuyau de plomb » comme clé de sélection.
On peut introduire dans une clé de sélection un ou plusieurs symboles à la place de caractères. Ces symboles sont les trois **signes de comparaison** :

< indique que le caractère situé à la même place dans la rubrique ne doit pas être testé, mais considéré comme supérieur à ce à quoi il aurait dû être comparé ;
> provoque l'effet inverse ;
= signifie que quel que soit ce caractère dans la rubrique, il doit être accepté (considéré comme égal à celui de la clé).

L'emploi de ces caractères est lié à la possibilité de sélectionner (ou d'exclure) un enregistrement, selon que la rubrique testée est inférieure, égale ou supérieure (LEG) à la clé introduite. Le fait d'utiliser ces symboles permet donc d'ignorer les caractères correspondants de la rubrique, en les rendant automatiquement inférieurs, égaux ou supérieurs à la clé, selon qu'on sélectionne suivant l'une ou l'autre de ces caractéristiques. Supposons, par exemple, que des composants soient répertoriés dans un fichier par un code général constitué en fait de codes différents :

Rubrique « code » du fichier des composants

Code de montage	Code du composant	Classe du composant
94	543678	B
67	553478	A
80	887690	A
88	113425	D
80	984400	C
80	000789	A
56	722990	B

On veut effectuer la sélection des composants de classe A et dont le code de montage est 80. Pour cela, il suffit d'utiliser la clé de sélection suivante :

80 = = = = = = A

Deux composants seront alors sélectionnés dans la partie du fichier reproduite :

80-887690-A
80-000789-A

Quand on utilise plusieurs clés de sélection/exclusion, on doit les relier par un opérateur logique AND ou OR.
Avec deux clés reliées par AND, un enregistrement doit remplir simultanément les deux conditions pour être sélectionné, tandis qu'avec OR, il suffit que l'une des deux soit satisfaite. Ainsi, en définissant pour le fichier de composants la combinaison suivante :

80 ═══ A OR 80 ═══ C

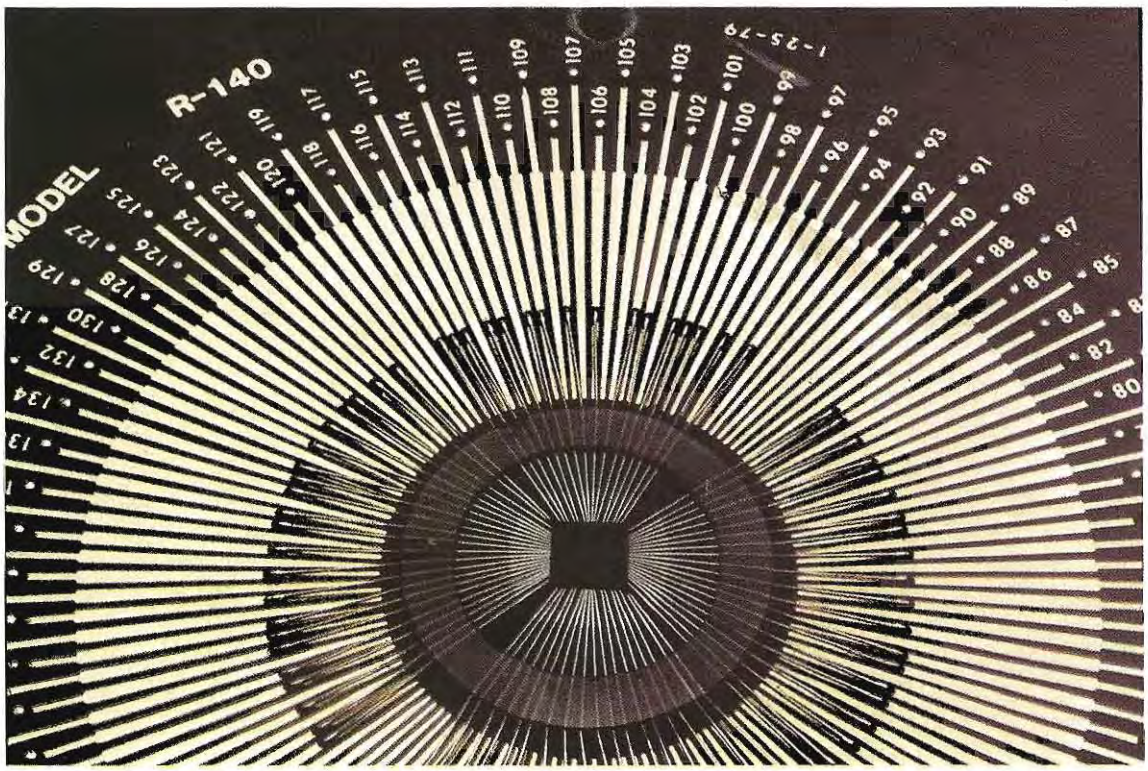
nous obtiendrons la sélection de trois composants :

80-887690-A
80-984400-C
80-000789-A

Notons que l'opérateur AND ne peut relier que des clés portant sur des rubriques différentes. Il serait en effet impossible de sélectionner un composant qui appartiendrait à la fois à classe A et à classe C.
Olisort permet également de regrouper deux fichiers en un seul. Deux types de combinaisons sont réalisables : la fusion (merge) qui opère sur deux fichiers classés et renvoie le fichier résultant ordonné, et la concaténation (append) simple qui n'effectue aucun tri.
La fusion nécessite donc une spécification des clés pour le tri.
Par contre, la concaténation de deux fichiers ne demande aucune précision particulière.
Olisort peut reconnaître jusqu'à cinq fichiers simultanément. Cependant, trois d'entre eux sont réservés au fonctionnement :

— INPUT 1 : fichier général du progiciel ;
— INPUT 2 : fichier utilisé lors d'une fusion ou d'une concaténation ;
— OUTPUT : fichier résultant de toute opération de tri, de fusion, de sélection ou de concaténation. Si on donne à ce fichier de sortie le nom du fichier original, il sera écrit sur le disque à la place de celui-ci.

Un testeur de circuits intégrés.



Les modalités d'exécution. Nous avons vu que la mise en œuvre du programme Olisort peut se faire :

- en fixant la valeur des paramètres de commande dans un programme, pour une exécution différée ;
- en passant par l'utilitaire Olisortx pour sélectionner en temps réel ces mêmes paramètres.

Les paramètres sont d'un format identique quelle que soit la procédure.
Ils peuvent être transmis à Olisort de deux façons :

- à l'aide d'un fichier paramètres sauvegardé sur disque (ce qui est avantageux dans le cas d'opérations d'usage fréquent). Un utilitaire Olisortp facilite la création de ce fichier ;
- par l'intermédiaire d'une suite de commandes regroupées au sein d'une chaîne alphanumérique. Cette chaîne s'appelle OLISORT.CMD$. et peut être créée à partir du Basic (programme) ou de l'Olisortx (direct). Dans ce second cas, elle sera mémorisable sur disque.

Notons que le recours à un fichier paramètres n'exclut pas l'insertion de chaînes de commande dans un programme. Ces chaînes seront cependant très brèves (un paramètre et une référence au fichier commandes).
L'un des paramètres permet de préciser le mode suivant lequel doit fonctionner le progiciel.
Le choix porte alors sur quatre modes.

Le **Mode 0** permet d'ordonner un fichier, de réaliser une sélection d'enregistrements ou d'effectuer conjointement ces deux opérations. Toutes les commandes nécessaires doivent avoir été préalablement mémorisées dans un fichier paramètres.
Supposons qu'on dispose d'un fichier permanent de stock classé selon les codes articles et de deux fichiers temporaires où sont répertoriés de nouveaux produits. On pourra procéder au tri de ces deux derniers fichiers de la façon suivante :

1 / création d'un fichier paramètres à l'aide de l'utilitaire Olisortp ;

2 / écriture d'un court programme Basic qui appellera deux fois Olisort afin d'exécuter le tri des deux fichiers selon le même principe (même fichier paramètres). Ce programme précisera le mode d'exécution d'Olisort (Mode 0) et la liste des commandes qui ne peuvent être intégrées au fichier paramètres.

Le **Mode 1** gère la fusion et la concaténation des fichiers. Ce mode se réfère lui aussi à un fichier paramètres. Si les deux fichiers sont ordonnés et que le fichier paramètres contient une clé de tri, Olisort procédera alors à une fusion. Dans le cas contraire, les deux fichiers seront simplement concaténés.
Reprenons les trois fichiers cités plus haut, on peut désormais effectuer la mise à jour du fichier permanent de la façon suivante :

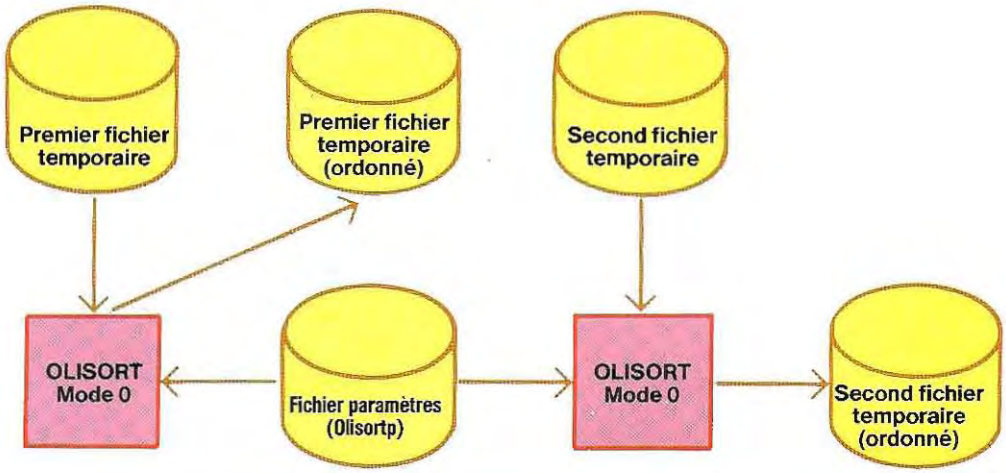
1 / on réutilisera le fichier paramètres créé pour le tri des fichiers temporaires mais on modifiera le programme Basic de façon à obtenir une exécution du mode 1. Les fichiers INPUT 1 et INPUT 2 seront les deux fichiers temporaires ordonnés ;
2 / l'exécution du programme Basic aboutira donc à la fusion de ces deux fichiers en un unique fichier temporaire ordonné lui aussi ;
3 / on reprendra alors le programme Basic précédent et on l'appliquera au nouveau fichier temporaire et au fichier permanent ;
4 / ce programme aura pour effet la fusion de ces deux fichiers réalisant ainsi une mise à jour du fichier permanent.

Le **Mode 2** sert à exécuter les mêmes opérations que le Mode 0 mais sans recours à un fichier paramètres. Les commandes devront donc être définies une à une et in extenso dans le programme Basic.

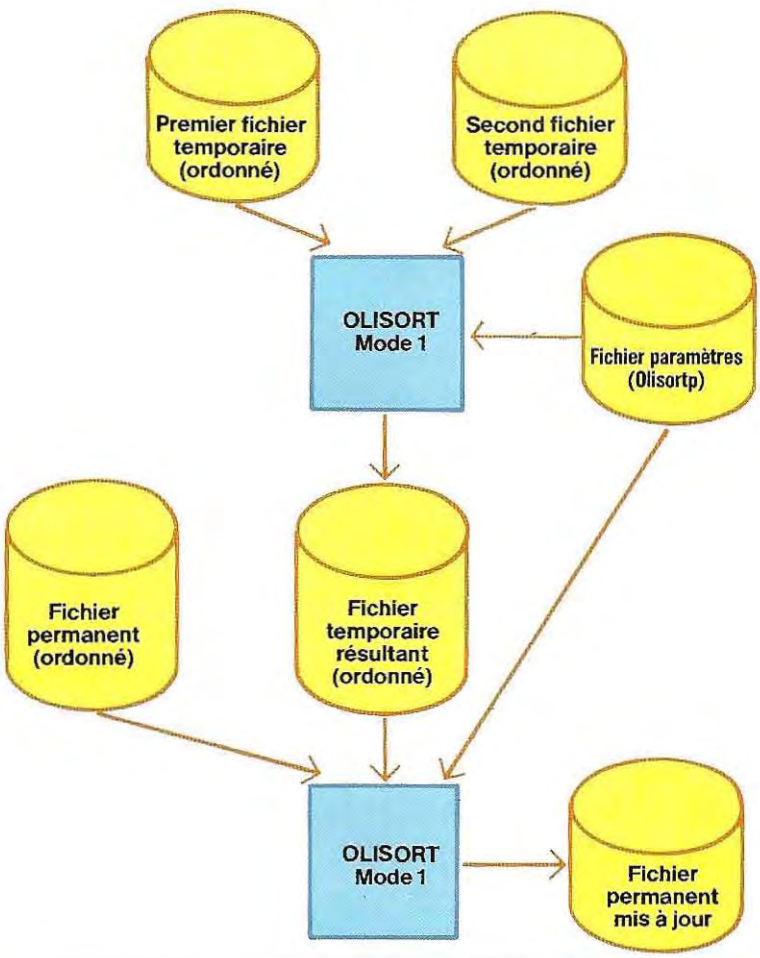
Le **Mode 3** est identique au Mode 2. Cependant, un fichier paramètres est créé à partir des commandes successives insérées dans le programme Basic et pourra donc être réutilisé par la suite.
Nous aurions ainsi pu réaliser le classement de nos deux fichiers temporaires de la façon suivante :

1 / en écrivant un programme Basic appelant Olisort pour le tri en Mode 3 du premier fichier temporaire ;

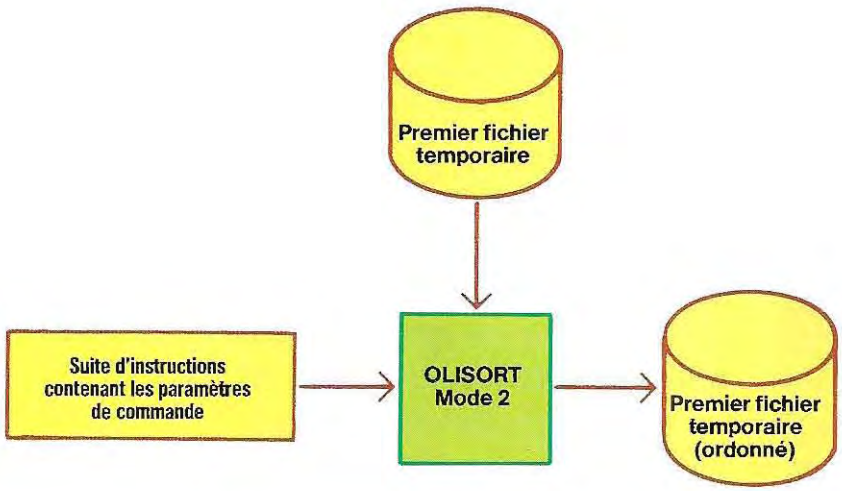
TRI EN MODE 0



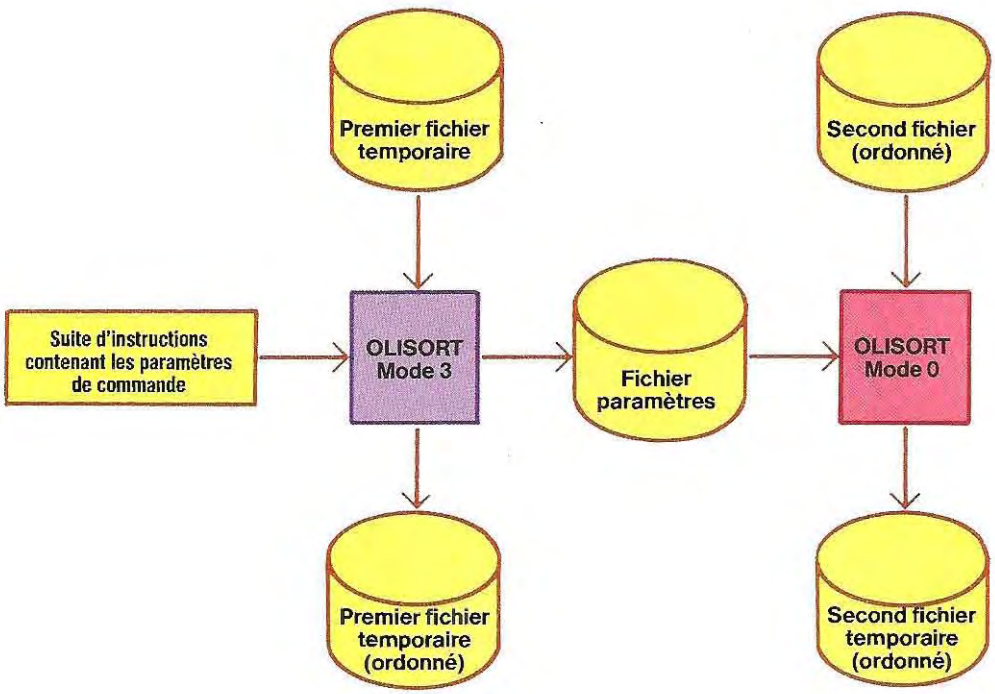
FUSION EN MODE 1



TRI EN MODE 2



TRI EN MODE 3



2 / l'exécution de ce programme aurait alors conduit non seulement au classement du fichier temporaire mais également à la sauvegarde des paramètres de commande dans un fichier paramètres ;

3 / ce dernier aurait ensuite pu être utilisé par un autre programme Basic pour effectuer en Mode 0 le classement du second fichier temporaire.

Conception d'un programme de gestion de données

Les ordinateurs sont généralement acquis pour mémoriser et traiter des données en quantité importante.

Sur les gros et les moyens systèmes, on dispose de programmes de gestion des données spécifiques, capables d'assurer la création et la maintenance des fichiers les plus complexes.

En revanche, les logiciels de gestion de données adaptés aux micro-ordinateurs se veulent universels et leur emploi impliquerait parfois des aménagements conséquents.

Il est alors préférable de rédiger soi-même un programme moins performant peut-être mais mieux adapté à un travail particulier.

Pour éviter d'écrire un programme spécifique à chaque application, on aura avantage à concevoir un logiciel bien structuré regroupant plusieurs modules paramétrés.

La modification de la valeur des paramètres offrira ainsi une plus large gamme d'utilisation. En règle générale, un programme de gestion de données est articulé sur les opérations suivantes :

- acquisition des données ;
- traitement ;
- mémorisation sur disque.

Il est préférable d'établir des sous-programmes gérant chacun une opération.

La rédaction d'un programme général en sera grandement simplifiée puisqu'il ne comportera guère que des renvois aux routines adaptées.

Il faudra, avant tout, spécifier le type et la longueur des variables à traiter.

Ces informations sont en effet indispensables aux sous-programmes de saisie et de mémorisation.

L'acquisition des données

Supposons qu'on veuille créer des enregistrements de la structure suivante :

Nom		9 caractères
Code		4 caractères
	Jour	2 caractères
Date	Mois	2 caractères
	Année	2 caractères

On devra définir un masque de saisie écran pour contrôler le nombre de caractères et le type de chaque donnée (ainsi la date doit être constituée exclusivement de caractères numériques). Toutefois, si une autre acquisition portant sur des données d'un autre type ou d'un autre format est nécessaire plus loin dans le programme, un masque différent devra être défini.

Préparation du masque de saisie

Les caractéristiques d'un masque de saisie sont définies par un certain nombre de données :

- ligne et colonne de début (position sur l'écran) ;
- nombre de lignes (chaque ligne correspondant à une entrée) ;
- libellé de chaque variable ;
- nombre de caractères de chaque variable ;
- type des variables.

Ainsi, dans l'exemple mentionné plus haut, nous aurions :

- position de début : ligne 4, colonne 6 (valeurs arbitraires) ;
- nombre de lignes ;
- libellés : 1 - Nom
  2 - Code
  3 - Jour
  4 - Mois
  5 - Année
- longueur de chaque variable :
  Variable 1 = 9 caractères
  Variable 2 = 4 caractères
  etc.
- type des variables
  Variable 1 = alphabétique
  Variable 2 = numérique
  etc.

Le rôle d'un masque de saisie est d'afficher le libellé suivi d'autant de points que de caractères à entrer, puis de positionner le curseur à l'emplacement du premier de ces caractères (coordonnées 6 et 4).
Les libellés peuvent être stockés dans une variable alphanumérique indicée. Dans notre exemple, cette variable devra être dimensionnée à cinq éléments. Si nous l'appelons LIB$(5), il suffira de demander l'affichage de LIB$(1) pour faire apparaître le premier libellé, de LIB$(2) pour le second et ainsi de suite. Pour les pointillés, le caractère « . » sera défini en temps que variable alphanumérique et utilisé pour former des chaînes de la longueur désirée. Ainsi, en posant A$=« . » et B$= STRING$(9,A$), on crée une chaîne B$ constituée de 9 points. On obtiendra la première ligne du masque de saisie par l'instruction :

```
PRINT LIB$(1)+" "+B$
```

L'espace sert à séparer le libellé des pointillés pour améliorer la présentation. L'emploi des instructions DATA et READ est le plus indiqué pour affecter les valeurs voulues aux paramètres du masque de saisie. On procèdera donc au remplissage de la table des libellés de la façon suivante :

```
10 RESTORE 30
20 FOR I=1 TO 5:READ LIB$(I):NEXT I
30 DATA "1 - Nom"
40 DATA "2 - Code"
50 DATA "3 - Jour"
60 DATA "4 - Mois"
70 DATA "5 - Année"
```

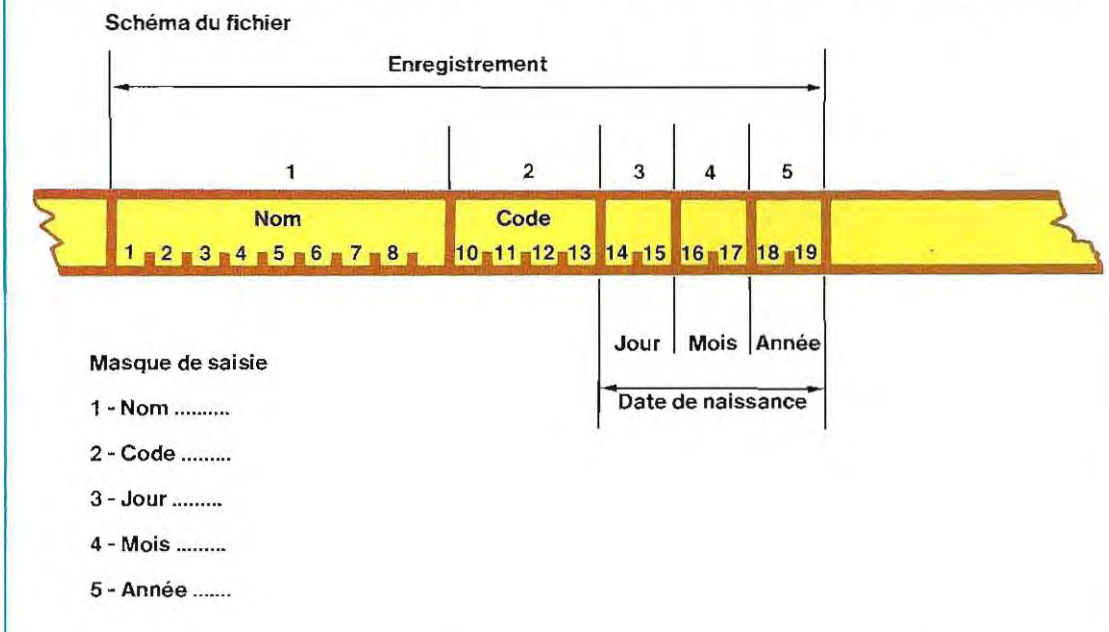
Dans le sous-programme de mémorisation, il faudra définir la longueur de chaque zone mais aussi sa position dans l'enregistrement : la zone « Nom » de l'exemple occupe les octets 1 à 9, la zone « Code » les octets 10 à 13, etc. Voici les noms qui ont été adoptés dans les programmes listés plus loin :

PO(I) = Premier octet de la variable I
DO(I) = Dernier octet de la variable I
TC(I) = Type de la variable I

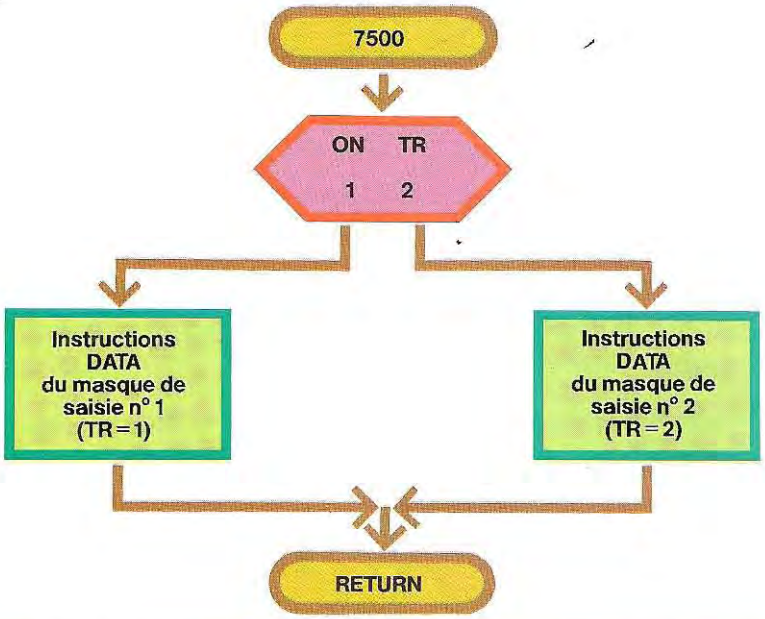
Dans le cas présent, il faudrait donner à ces variables les valeurs suivantes :

PO(1) = 1 DO(1) = 9 (première zone, octets 1 à 9) ;
PO(2) = 10 DO(2) = 13 (seconde zone, octets 10 à 13) ;
PO(3) = 14 DO(3) = 15 (troisième zone, octets 14 à 15).

STRUCTURE D'UN ENREGISTREMENT ET MASQUE DE SAISIE CORRESPONDANT



SOUS-PROG. D'AFFECTATION DES LIBELLES ET DES LONGUEURS DE CHAMPS



On définira de la même façon le type de chaque champ. A cet effet, on adoptera, par exemple, la convention suivante :

TC(I) = 1 pour une variable numérique ;
TC(I) = 2 pour une variable alphanumérique ;
TC(I) = 3 pour une variable faisant l'objet d'une simple présentation.

La première zone de notre exemple sera alors de type 2, tandis que toutes les autres seront de type 1. On utilise le type 3 pour définir des champs dont le contenu doit être visualisé sans que l'opérateur puisse le modifier.
Certaines applications nécessitent l'affichage de plusieurs masques de saisie successifs : c'est le cas du programme que nous allons maintenant développer. Il sera judicieux de réutiliser les mêmes variables pour chacun des masques. Il suffit alors de prévoir plusieurs groupes d'instructions DATA et d'activer sélectivement celui qui contient les données voulues. On peut, au niveau du programme d'appel, mémoriser le numéro du masque à visualiser dans une variable (appelée par exemple NM), dont la valeur déterminera, dans le sous-programme, le choix des instructions d'affectation correspondantes. Le sous-programme définit également le nom et le numéro du fichier dans lequel les données doivent être écrites (représentés respectivement par les variables NF$ et NO).
L'organigramme du haut de cette page correspond à un sous-programme dans lequel on n'a prévu que deux masques de saisie (NM peut prendre la valeur 1 ou 2). Pour lui conférer un domaine d'application plus large, il suffirait d'ajouter des groupes de DATA, des ordres de lecture et, bien sûr, de modifier en conséquence l'instruction de branchement multiple. Cette routine est listée page 698.
La structure de chaque nouveau groupe de DATA devra se calquer sur celle du bloc compris entre les lignes 7800 et 7824.
La première instruction (ligne 7530) sert à définir la position du masque à visualiser (KC=2, KL=3) ; l'exécution se poursuit ensuite en fonction du numéro de masque choisi (ligne 7540).
Dans chaque cas, on initialise tout d'abord les variables caractéristiques du masque : son nombre de champs (NC) et le numéro logique du fichier associé (NO). On positionne ensuite le pointeur de DATA, et on appelle la routine 7730, qui effectuera toutes les lectures et affectations (ces instructions sont en effet communes à tous les masques). Au retour, on mémorise le nom du fichier dans

AFFECTATION DES LIBELLÈS ET DES LONGUEURS DE CHAMPS

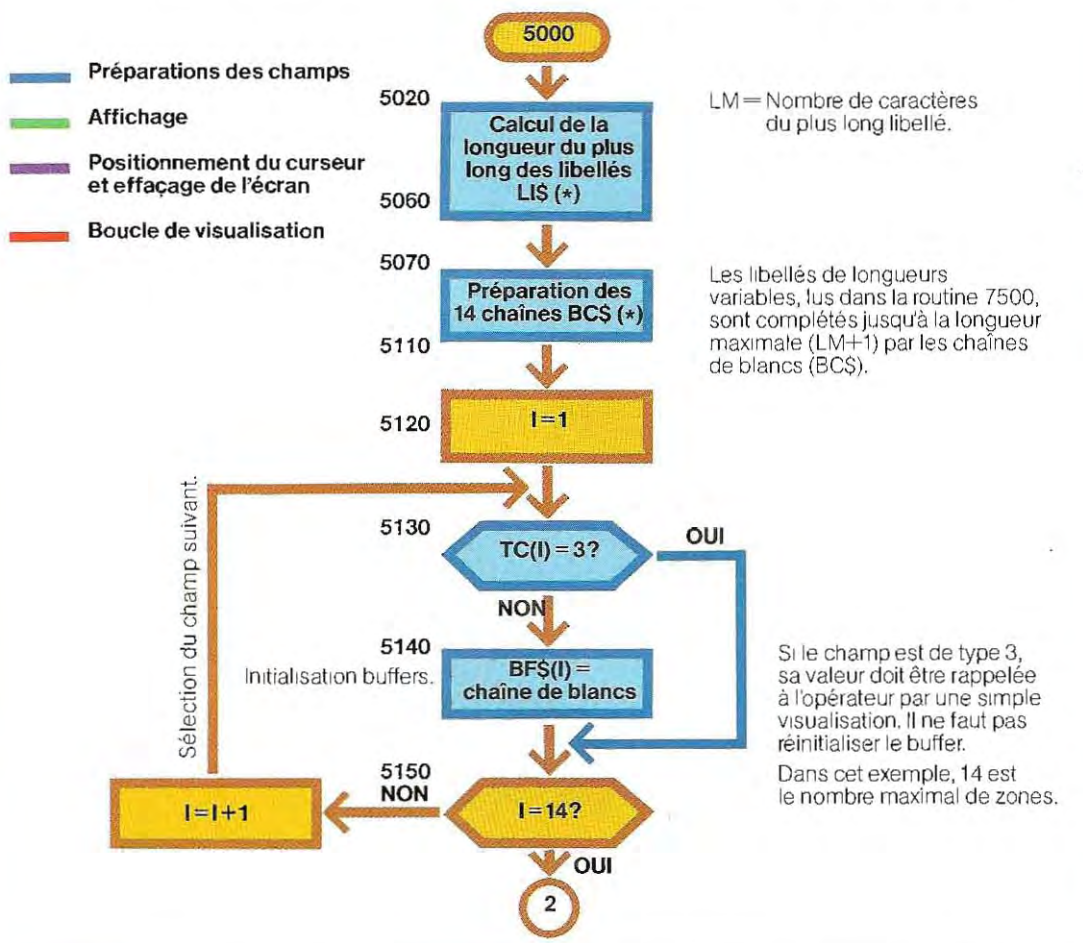
```
7500 '**SOUS-PROGRAMME DATA**
7502 '   FICHIER = ENCYCL
7504 '
7506 'ENTREE :    NM = numero de masque
7508 'SORTIES:    KC = position de la premiere colonne sur l'ecran
7510 '            KL = position de la premiere ligne
7512 '         NC(*) = nombre de champs
7514 '       LI$(*) = libelle des lignes
7516 '         TC(*) = type de champ
7518 '         PO(*) = pointeur vers le premier octet du champ
7520 '         DO(*) = pointeur vers le dernier octet
7522 '         LO : NO = numero logique du fichier
7524 '           NF$ = nom du fichier
7526 '           LN = nombre de caracteres d'un enregistrement du fichier
7528 '
7530 KC = 2 : KL = 0    'Position du curseur
7540 ON NM GOTO 7570, 7640    'Selection selon le numero de masque
7550 '
7560 '* SI NM = 1
7570 NC(NM) = 10 : N = 10 : NO = 1    'Initialisations
7580 RESTORE 7800                     'Positionnement dans les DATA
7590 GOSUB 7740                       'Lecture des DATA
7600 NF$ = "A:ACHVEN"                 'Nom du fichier
7610 GOTO 7690
7620 '
7630 '* SI NM = 2
7640 NC(NM) = 9 : N = 9 : NO = 2
7650 RESTORE 7830
7660 GOSUB 7740
7670 NF$ = "A:MVTMAR"
7680 '
7690 N = N+1                          'Reinitialisation des champs excedentaires
7700 FOR I = N TO 14 : LI$(I) = "" : TC(I) = 0 : PO(I) = 0 : DO(I) = 0 : NEXT I
7710 RETURN                           'Sortie du sous-programme DATA
7720 '
7730 '**LECTURE DES DATA
7740 FOR I = 1 TO N : READ LI$(I) : NEXT I
7750 FOR I = 1 TO N : READ TC(I) : NEXT I
7760 FOR I = 1 TO N : READ PO(I) : NEXT I
7770 FOR I = 1 TO N : READ DO(I) : NEXT I
7780 LN = DO(N)
7790 RETURN                           'Retour au sous-programme DATA
7795 '
7796 '**DATA
7798 '* cas ou NM = 1
7800 DATA "  1 - Type marchandise"
7802 DATA "  2 - Taux T.V.A."
7804 DATA "  3 - Unite de mesure"
7806 DATA "  4 - Cout Achat"
7808 DATA "  5 - Prix Vente"
7810 DATA "  6 - Cumul des Entrees"
7812 DATA "  7 - Valeur totale"
7814 DATA "  8 - Cumul des Sorties"
7816 DATA "  9 - Valeur totale"
7818 DATA "10 - ..........."
7820 DATA 2!, 1!, 2!, 2!, 2!, 3!, 3!, 3!, 3!, 3!
7822 DATA 1!, 21!, 23!, 31!, 38!, 45!, 53!, 63!, 72!, 82!
7824 DATA 20!, 22!, 30!, 37!, 44!, 52!, 62!, 71!, 81!, 100!
7826 '* cas ou NM = 2
7830 DATA " 1 - CODE MARCH."
7832 DATA " 2 - DATE MOUVENENT"
7834 DATA " 3 - QTE ENTREE"
7836 DATA " 4 - COUT ACHAT"
7838 DATA " 5 - QTE SORTIE"
7840 DATA " 6 - PRIX VENTE"
7842 DATA " 7 - ..........."
7844 DATA " 8 - ..........."
7846 DATA " 9 - transfert Mvt."
7848 DATA 1!, 2!, 2!, 2!, 2!, 2!, 2!, 2!, 3!
7850 DATA 1!, 4!, 9!, 17!, 24!, 32!, 38!, 65!, 91!
7852 DATA 3!, 8!, 16!, 23!, 31!, 38!, 64!, 90!, 91!
```

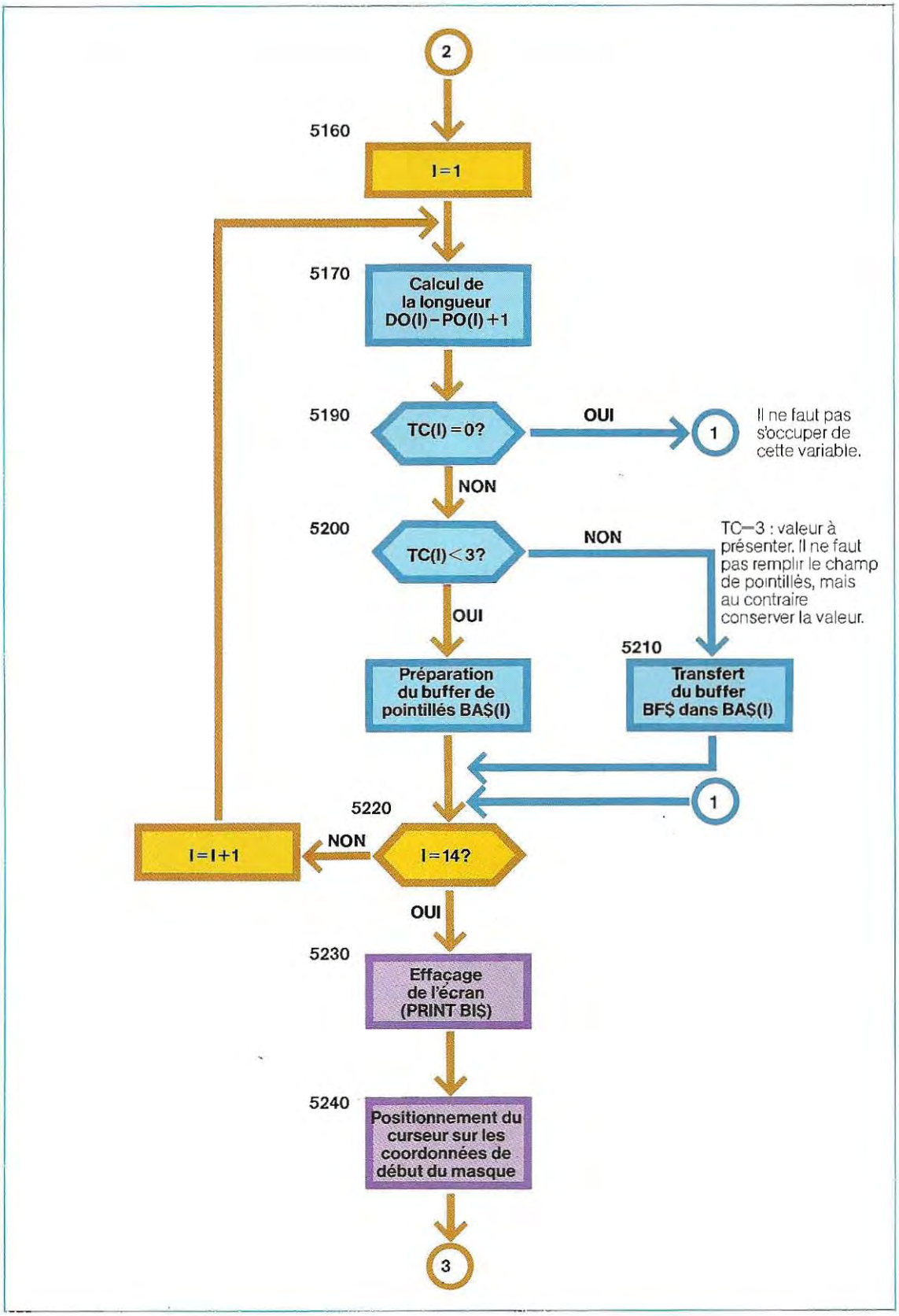
la chaîne NF$. Dans ce programme, le fichier correspondant au premier masque s'appelle ACHVEN et se trouve sur la disquette A.
Enfin, les dernières lignes (7690-7700) réinitialisent les éléments inutilisés des tableaux. Ainsi, avec le masque 1, on n'affiche que 10 zones, alors que le maximum prévu est 14. Les variables excédentaires sont mises à zéro ou à vide, selon leur type, dans une boucle qui commence à N+1, N étant le nombre de lectures effectivement exécutées (ligne 7570 pour le masque 1).

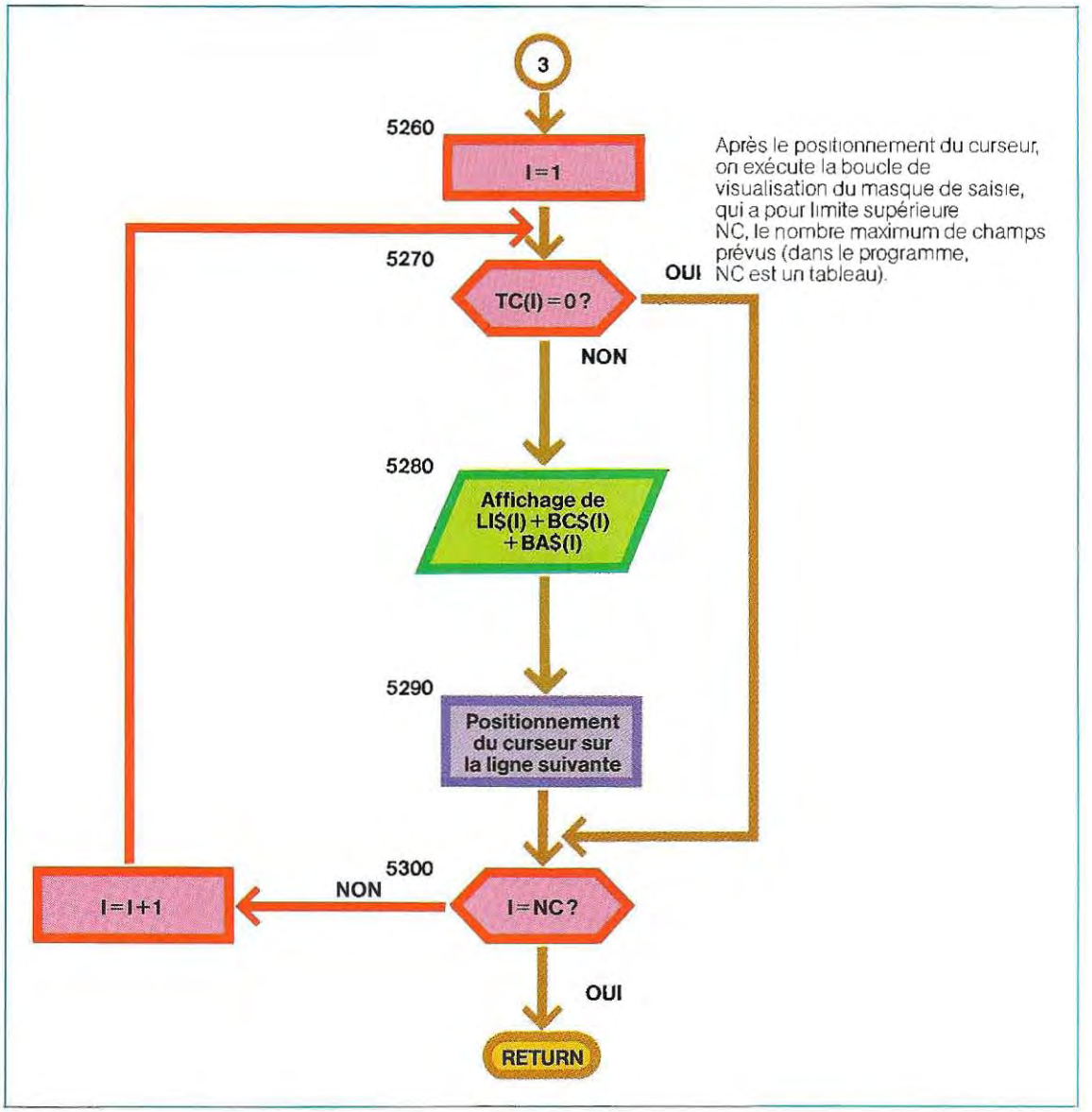
Affichage du masque de saisie. Une fois qu'on a appelé le sous-programme précédent, il faut visualiser le masque de saisie. Cette opération présente deux difficultés : le cadrage des zones et, éventuellement, la présence de variables de type 3.

Les différents libellés ne possèdent pas forcément la même longueur. Il ne faut donc pas afficher systématiquement les pointillés à la suite du libellé correspondant, car ils ne seraient alors pas alignés les uns avec les autres. Pour éviter cette mauvaise présentation, on procédera de la façon suivante : on cherchera la longueur du plus grand libellé, et on complètera chaque libellé par un nombre de blancs égal à la différence entre la longueur maximale et celle du libellé lui-même. En fait, il vaudra mieux prévoir chaque fois un espace supplémentaire, afin que les pointillés ne soient pas accolés au libellé le plus long.
L'organigramme du sous-programme de visualisation des masques de saisie commence page 699 et se poursuit pages 700 et 701. Le listing correspondant se trouve page 702.

SOUS-PROGRAMME D'AFFICHAGE DES MASQUES DE SAISIE




2
5160
I=1
5170
Calcul de la longueur DO(I)-PO(I)+1
5190
TC(I)=0?
OUI
1
Il ne faut pas s'occuper de cette variable.
NON
5200
TC(I)<3?
NON
TC=3 : valeur à présenter. Il ne faut pas remplir le champ de pointillés, mais au contraire conserver la valeur.
OUI
5210
Préparation du buffer de pointillés BA$(I)
Transfert du buffer BF$ dans BA$(I)
1
5220
I=I+1
NON
I=14?
OUI
5230
Effacage de l'écran (PRINT BI$)
5240
Positionnement du curseur sur les coordonnées de début du masque
3



Certaines des variables mentionnées dans ce sous-programme sont initialisées dans le programme principal, et notamment :

BL$ = chaîne d'espaces blancs
BP$ = chaîne de pointillés

La recherche du nombre de caractères du plus long libellé s'effectue par les instructions 5020 à 5060, tandis que la boucle suivante (5070 à 5110) construit les chaînes BC$ de blancs, qui viendront compléter les différents libellés LI$. Les chaînes BC$ sont générées par prélèvement de N caractères dans BL$ (instruction 5100).

Deux formes de présentation sont prévues dans le sous-programme : l'une pour les valeurs à saisir au clavier et l'autre pour celles qui doivent seulement être affichées. En effet, le type de champ (TC), lu dans le sous-programme 7500 prend les valeurs 1, 2 ou 3. Dans les deux premiers cas, on affichera une chaîne de pointillés (BA$). Pour le type 3, au contraire, la chaîne BA$ à afficher devra contenir la valeur lue et affectée au buffer BF$. Ainsi, pour obtenir la visualisation d'un masque de saisie, on appellera successivement le module 7500 qui contient les DATA, puis le module 5000. Pour visualiser le premier masque, on exécutera les instructions suivantes :

AFFICHAGE DES MASQUES DE SAISIE

```
5000 ' AFFICHAGE DES MASQUES DE SAISIE
5002 ' ** sous-programme visuel **
5004 '        FICHIER = ENCYCL
5006 '
5008 ' ENTREES : LI$(*),TC(*),PO(*),DO(*),NC(*),KC,KL de la routine 7500
5010 '           BF$(*)                                 de la routine 1200
5012 '           NM                                     numero de masque
5014 ' SORTIE  : LO(*)=longueur des champs
5016 '           BF$(*)=buffer des champs, corrige a la bonne longueur
5020 LM=0
5030 FOR I=1 TO 14                  'Recherche de la longueur maximale
5040 N=LEN(LI$(I))                  ' des libelles
5050 IF N>LM THEN LM=N
5060 NEXT I
5070 '
5080 FOR I=1 TO 14                  ' Chaines de blancs pour completer
5090 N=LEN(LI$(I))                  ' les libelles
5100 NB=LM-N+1
5110 BL$(I)=LEFT$(BL$,NB)
5114 NEXT I
5115 '
5120 FOR I=1 TO 14
5130 IF TC(I)=0 OR TC(I)=3 GOTO 5150
5140 BF$(I)=BL$                     ' Reinitialisation du buffer
5150 NEXT I
5155 '
5160 FOR I=1 TO 14
5170 N=PO(I)-PO(I)+1                ' Longueur du champ
5180 LO(I)=N
5190 IF TC(I)=0 GOTO 5220
5200 IF TC(I)<3 THEN BA$(I)=LEFT$(BF$,N) : BF$(I)=LEFT$(BF$(I),N) : GOTO 5220
5210 BA$(I)=BF$(I)                  ' Transfert de la donnee a afficher
5220 NEXT I
5225 '
5230 PRINT BI$          ' Efface l'ecran et emet un bip
5235 '
5240 X=KC : Y=KL : GOSUB 6900       ' Position curseur
5250 NE=NC(NM)                  ' Nombre de lignes a afficher
5260 FOR I=1 TO NE
5270 IF TC(I)=0 GOTO 5300
5280 PRINT LI$(I)+BC$(I)+BA$(I)
5290 Y=Y+1 : GOSUB 6900             Incrementation d'une ligne
5300 NEXT I
5310 RETURN
```

NM=1
GOSUB 7500
GOSUB 5000

Dans le module 7500, les cinq dernières zones du premier masque sont définies en type 3. Or, ces champs n'ont pas encore fait l'objet d'une affectation. Ils peuvent donc être vides, ou contenir des valeurs erronées provenant de traitements antérieurs. Si l'on veut afficher des valeurs correctes, il faut les initialiser avant d'appeler le sous-programme 5000. Ces zones portent les numéros 6, 7, 8, 9 et 10. Leurs libellés figurent aux lignes 7810 à 7818, et voici leurs longueurs respectives :

champ 6	de 45 à 52 = 8 caractères
champ 7	de 53 à 62 = 10 caractères
champ 8	de 63 à 71 = 9 caractères
champ 9	de 72 à 81 = 10 caractères
champ 10	de 82 à 100 = 19 caractères

Si l'on transfère des chaînes de caractères dans les buffers BF$ correspondants, celles-ci seront visualisées en même temps que le reste du masque de saisie. Ecrivons, par exemple :

NM=1
GOSUB 7500
BF$(9)="0123456789"
GOSUB 5000

L'exécution de ces instructions entraîne l'affichage de la chaîne "0123456789" sur la neuvième ligne du masque.

Entrée des données. Après le module 5000, on parvient en phase de saisie des données. Les opérations à exécuter sont les suivantes :

— positionnement du curseur ;
— lecture d'un caractère (sans écho) ;
— contrôle du caractère lu.

La suite des opérations dépend de l'issue de ce contrôle. Quatre cas peuvent se présenter :

1 - le caractère entré est un caractère de commande ;
2 - il n'est pas homogène avec le type de variable attendu ;
3 - il est homogène et donc accepté ;
4 - il n'est pas reconnu.

Dans le deuxième et le quatrième cas, il faut repositionner le curseur sur le même point et demander une nouvelle entrée (correction). Dans le troisième cas, le caractère est transféré dans le buffer de la variable ; si, par exemple, le curseur se trouve sur la ligne 4, le caractère est mis dans le quatrième buffer. Enfin, dans le premier cas, la machine doit exécuter les fonctions associées au caractère de commande.
Quand on conçoit un masque de saisie, il faut toujours offrir à l'opérateur la possibilité d'effectuer des corrections. Les déplacements du curseur sont commandés par quatre touches programmées dont la frappe déclenche l'envoi d'un code numérique spécial. Le tableau TF, qui regroupe les différents codes de commande, est initialisé dans le programme principal. Pour déterminer si la touche frappée est associée à une fonction, il suffit de balayer ce tableau. Si le nombre généré s'y trouve, on va se brancher à un endroit du programme défini par sa valeur (lignes 6450-6462). Ici, les codes associés aux touches programmées sont 10, 8, 9 et 11 (pour les déplacements), ainsi que 16, 17 et 13. Ces touches de fonction se répartissent en deux groupes : le premier commande des fonctions internes au masque de saisie et le second permet de sortir du sous-programme. La gestion des touches du premier groupe présente quelques difficultés. Prenons, par exemple, le cas des deux touches de déplacement vertical. Voici les opérations qu'elles réalisent et les codes correspondants :

— déplacement d'une ligne vers le haut, code 11 (ligne 6590) ;
— déplacement d'une ligne vers le bas, code 10 (ligne 6470).

On peut résumer ainsi les opérations effectuées par ces touches :

— incrémentation (ou, selon le sens du déplacement, décrémentation) du compteur de lignes CL ;
— incrémentation (ou décrémentation) de la coordonnée verticale Y ;
— remise à zéro du compteur de caractères (CC) ;
— positionnement horizontal en début de zone (X=X0).

Supposons, par exemple, que le curseur se trouve sur la ligne 3 (CL=3). Les données qu'on entre à ce moment sont transférées dans le buffer numéro 3 (BF$(3)). L'activation du code 11 devra entraîner :

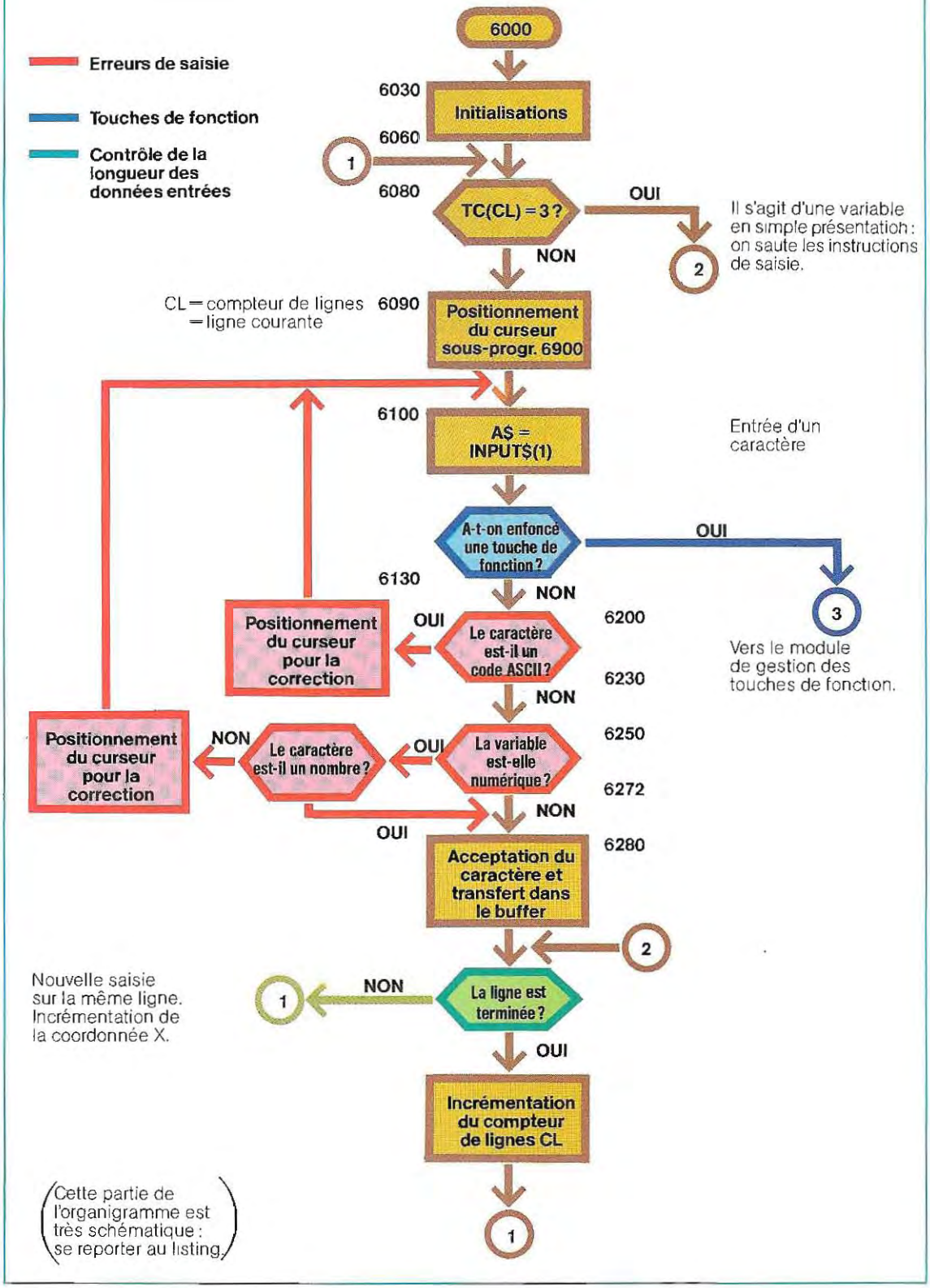
— la décrémentation du compteur de lignes (CL=3-1=2) qui doit désormais pointer vers le buffer numéro 2 ;
— la décrémentation de la position (Y=Y-1) pour que le curseur revienne sur la ligne précédente ;
— la remise à zéro du compteur de caractères (CC=0). Ainsi, le curseur ira se placer au début de la zone réservée à la donnée, et tout se passera comme si aucun caractère n'avait encore été transféré dans le buffer numéro 2.

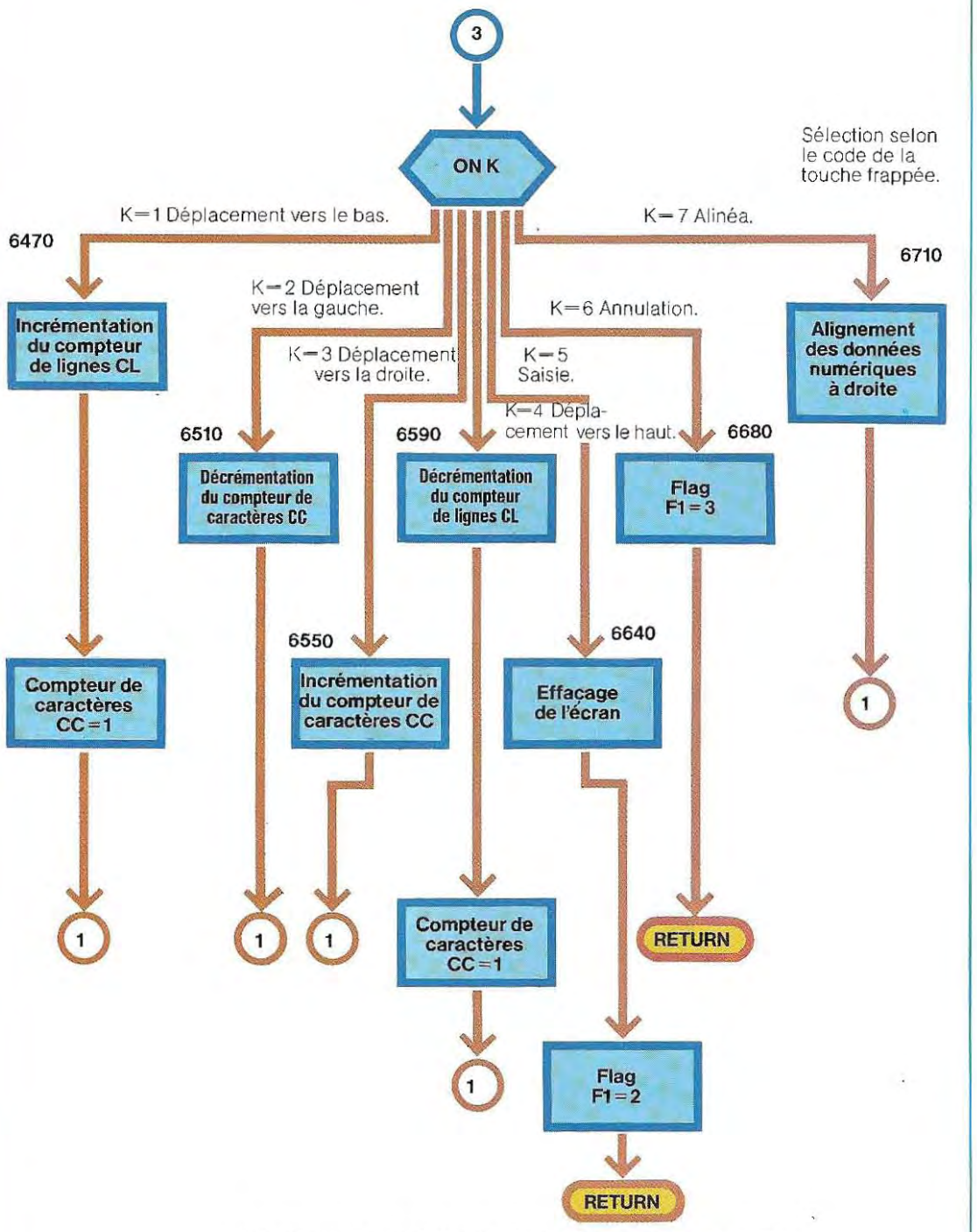
Les autres touches de fonction internes commandent les opérations suivantes :

— déplacement d'une position vers la droite, code 9, ligne 6550 ;
— déplacement d'une position vers la gauche, code 8, ligne 6510 ;
— justification, code 13, ligne 6710 (touche RT).

Le mécanisme des fonctions de déplacement horizontal est semblable à celui des fonctions de déplacement vertical, mais le compteur de lignes reste inchangé, alors que le compteur de caractères est incrémenté (ou décrémenté).

SOUS-PROGRAMME D'UTILISATION DES MASQUES DE SAISIE





On remarquera qu'il n'est possible de sortir du sous-programme qu'en frappant les touches de fonction associées à la saisie ou à l'annulation. Selon le cas, le flag est positionné à 2 ou à 3. Sa valeur est ensuite testée dans le programme principal, où elle sert d'aiguillage vers la saisie (2) ou l'annulation suivie d'une nouvelle saisie (3).

UTILISATION DES MASQUES DE SAISIE

```
'
' UTILISATION DES MASQUES DE SAISIE
'
' ** SOUS PROGRAMME SAISIE **
'    FICHIER = ENCYCL
'
' ENTREES :      TC(*),KC,KL          de la routine 7500
'                LM,LC(*),BF$(*)      de la routine 5000
'                NM = numero du masque
'                TF(20) = codes des touches programmées (pr. principal)
'    code        ASCII       Signification standard     fonction programmée
'     10         CTRL+J      line feed                  curseur bas
'      8         CTRL+H      backspace                  curseur gauche
'      9         CTRL+I      horizontal tabulation      curseur droite
'     11         CTRL+K      vertical tabulation        curseur haut
'     16         CTRL+P      DEL                        saisie des donnees
'     17         CTRL+Q      DCA                        annulation des donnees
'     13         CTRL+M      carriage return            justification
' SORTIES :   FI=flag  saisie/annulation
'
' CL = compteur de lignes = ligne courante
' CC = compteur de caractères
'
'
X0=KC+LM+1 : Y0=KL
CL=1 : CC=0
FI=0                    ' Flag de saisie/annulation
X=X0 : Y=Y0
'
IF TC(CL)=0 THEN GOSUB 6850 ' Ligne de masque inexistante : on cherche la suivante
IF TC(CL)=3 GOTO 6400       ' Valeur affichee : pas de saisie
GOSUB 6900                  ' Position du curseur en (X,Y)
'
A$=INPUT$(1)                ' Saisie d'un caractere sans echo
GOSUB 6500
U=ASC(A$)                   ' Conversion
'
K=0                         ' La touche frappée est-elle programmee?
FOR I=1 TO 20
IF TF(I)=0 GOTO 6170
IF U=TF(I) THEN K=I
NEXT I
IF K<>0 GOTO 6440           ' Gestion des touches de fonction
PRINT A$
'
' Controle du caractere frappé
IF U<123 AND U>31 GOTO 6240     ' Caractere ASCII
PRINT CHR$(7) : GOSUB 6850      ' On signale l'erreur a l'operateur
GOTO 6090                       ' Vers une nouvelle saisie
'
' Controle du type de caractere
IF TC(CL)=2 GOTO 6280           ' Caractere alpha
IF U>47 AND U<58 GOTO 6280      ' Caractere numerique correct
PRINT CHR$(7)
GOTO 6090
'
' Caractere accepte : insertion
N=LC(CL)                        ' Longueur totale du champ
NB=LEN(BF$(CL))                 ' Nombre de caracteres deja memorises
A$=INKEY$                       ' Caractere en cours de saisie
BF$(CL)=LEFT$(BF$(CL),NB)
ND=N-NB                         ' Nombre de caracteres encore a saisir
IF ND=0 THEN BD$="" : GOTO 6360
BD$=RIGHT$(BF$(CL),ND)
BF$(CL)=BD$+A$+BF$
```

```
6370 IF CC=N THEN CL=CL+1 : CC=0 : Y=Y+1 : X=X0-1      ' La ligne est complete
6380 IF CL>NC(NM) THEN CL=1 : Y=Y0                     ' Retour en debut de masque
6390 Y=Y+1                                            ' Deplacement vers la droite
6392 GOTO 6070
6395 '
6400 ' Cas d'une valeur affichee a ne pas modifier (TC(CL)=3)
6405 '
6410 CL=CL+1 : Y=Y+1              ' On passe simplement a la ligne
6420 IF CL>NC(NM) THEN CL=1 : Y=Y0 : CC=0 : X=X0      ' Retour en debut de masque
6430 GOTO 6070
6434 '
6438 '
6440 ' ** Gestion des touches programmees **
6445 '
6450 ON K GOTO 6470, 6510, 6550, 6590, 6640, 6680, 6710
6455 '
6456 ' Erreur : code fonction non attribue
6460 ER=5 : PU=5
6462 RETURN          ' Sortie du sous-programme
6465 '
6470 ' * U=10        deplacement vers le bas
6480 CL=CL+1 : CC=0
6485 X=X0 : Y=Y+1
6490 IF CL>NC(NM) THEN CL=1 : Y=1        ' Retour en debut de masque
6500 GOTO 6070
6505 '
6510 ' * U=8         deplacement vers la gauche
6520 IF CC=0 THEN GOSUB 6950 : GOTO 6090      ' On ne fait rien
6530 CC=CC-1 : X=X-1
6540 GOTO 6090
6545 '
6550 ' * U=9         deplacement vers la droite
6560 IF CC>LC(CL) THEN GOSUB 6950 : GOTO 6090
6570 CC=CC+1 : X=X+1
6580 GOTO 6090
6585 '
6590 ' * U=11        deplacement vers le haut
6600 IF CL=1 THEN GOSUB 6950 : GOTO 6090
6610 CL=CL-1 : Y=Y-1
6615 CC=0 : X=X0
6620 IF TC(CL)=0 GOTO 6600
6625 ' Si la ligne du masque n'est pas valide : on remonte encore d'une ligne
6626 ' Lorsqu'on se deplace en descendant, la recherche d'une ligne valide
6627 ' s'effectue dans la subroutine 6850
6630 GOTO 6080
6635 '
6640 ' * U=16        touche de saisie
6650 PRINT BI$
6660 F1=2                    ' Flag de saisie
6670 RETURN                  ' Sortie du sous-programme
6675 '
6680 ' * U=17        touche d'annulation
6690 F1=2                    ' Flag d'annulation
6700 RETURN                  ' Sortie
6705 '
6710 ' * U=13        justification a droite des donnees numeriques
6720 IF TC(CL)=2 GOTO 6810          ' Donnee alpha
6730 A$=""
6740 N=LC(CL)-CC             ' Nombre de zeros a ajouter a gauche
6750 IF N=0 GOTO 6810
6760 FOR I=1 TO N
6764 A$=A$+"0"
6767 NEXT I
6770 X=X0 : GOSUB 6900
6780 IF CC<>0 THEN A$=A$+LEFT$(BF$(CL),CC)
6790 PRINT A$
6800 BF$(CL)=A$
6805 '
```

```
6810  CL=CL+1 : CC=0
6815  X=X0 : Y=Y+1
6820  IF CL>NC(CL) THEN CL=1 : Y=Y0
6830  GOTO 6870
6835  '
6840  '
6845  '
6850  ' ** Routine de recherche d'une ligne de masque valide **
6855  '
6860  CC=0 , Y=X0
6870  CL=CL+1
6880  IF CL>NC(CL) THEN CL=1 : Y=Y0        ' Retour en debut de masque
6890  IF TC(CL)=0 GOTO 6870                ' On continue la recherche
6895  RETURN
6896  '
6897  '
6898  '
6900  ' ** Routine de positionnement du curseur **
6905  '
6910  PRINT CHR$(27)+CHR$(61)+CHR$(31+Y)+CHR$(31+X);
6920  RETURN
6925  '
6930  '
6940  '
6950  ' ** Routine de signalement des erreurs **
6960  '
6970  PRINT CHR$(7)
6980  PRINT CHR$(27)+CHR$(61)+CHR$(51)+CHR$(51);    ' Curseur en 20,20
6990  INPUT "ERREUR : tapez un caractere";S$
6995  PRINT CHR$(27)+CHR$(61)+CHR$(51)+CHR$(51); : PRINT " 30 "
6996  RETURN
```

Quant à la touche RT, on l'utilise pour aligner les données. En effet, lors de la saisie, le curseur se place au début de chaque zone, puis se décale d'une position vers la droite chaque fois qu'un caractère est saisi. Les données s'alignent donc toutes à gauche, quel que soit leur type, alors que les valeurs numériques doivent être cadrées à droite.

Les instructions 6710 à 6830 testent la nature de la variable (numérique ou alphanumérique selon la valeur de TC) et, éventuellement, rectifient l'alignement en ajoutant une chaîne de zéros à gauche du champ numérique. Elles sont activées par la frappe de la touche de justification.

Enfin, la fonction de positionnement du curseur est traitée séparément, dans un sous-programme d'une seule ligne (6900).

Quant aux fonctions du deuxième groupe, nous avons vu qu'elles permettent de sortir du module de visualisation. Au nombre de deux, elles se différencient par la valeur que prend le flag F1, selon qu'on frappe la touche associée à l'une ou à l'autre. On peut ainsi savoir, dans le programme principal, si les données sont correctes (touche 1, F1=2) ou si l'opérateur a décidé leur annulation (touche 2, F1=3).

Pour définir d'autres fonctions en programmant de nouvelles touches, il suffit de faire figurer leur code dans le tableau TF et d'ajouter au sous-programme de saisie les instructions nécessaires.

La phase de mémorisation sur disque

Lorsque la condition F1=2 est vérifiée (données correctes), les données doivent être écrites sur le disque. On peut encore concevoir un sous-programme d'emploi général.

Voici les quatre opérations qu'on peut effectuer sur un fichier :

- ouverture et définition de la zone tampon pour E/S ;
- lecture des enregistrements ; ,
- écriture des enregistrements ;
- fermeture du fichier.

Une instruction non paramétrée (CLOSE n) suffit à fermer un fichier. Il vaut donc mieux la placer dans le programme principal, plutôt que l'inclure dans la routine.

Les trois autres fonctions sont, au contraire, paramétrées. On devra donc, avant d'appeler

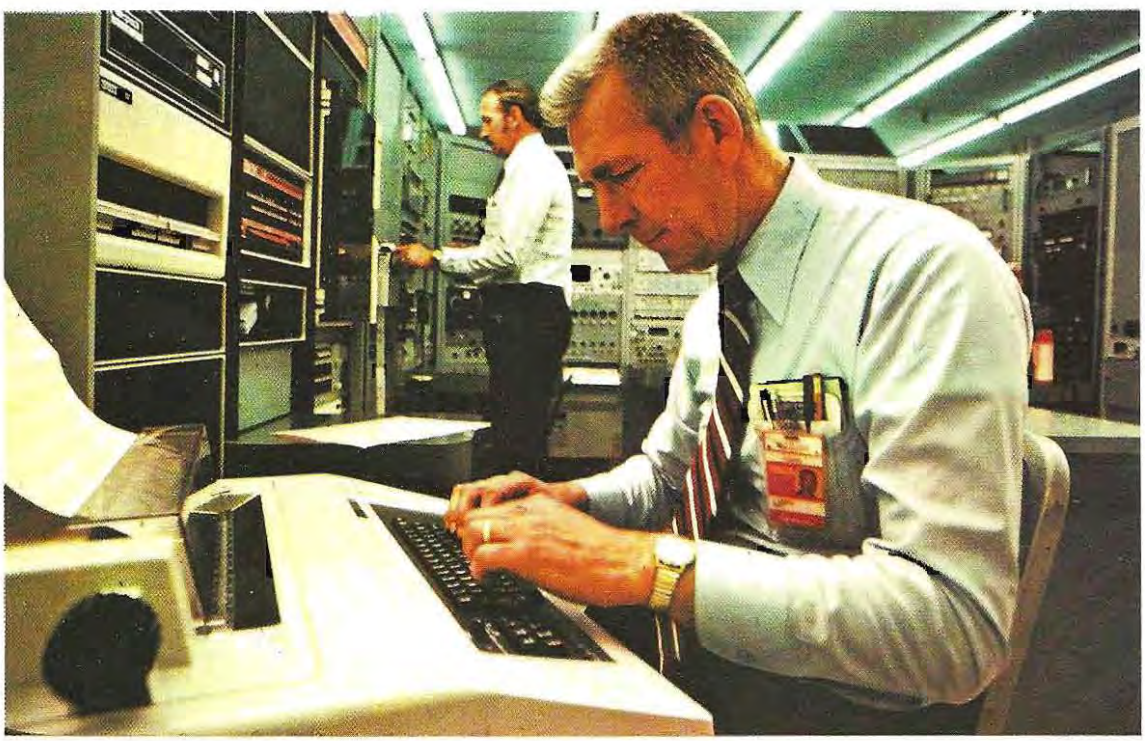
le sous-programme 1200 de gestion du disque, définir les paramètres suivants :

OP = flag indiquant l'opération désirée (ouverture, lecture, écriture) ;
NF$ = nom du fichier à traiter ;
NO = numéro d'unité logique de ce fichier ;
NE = numéro d'enregistrement, en lecture ou en écriture ;
LN = longueur de cet enregistrement (tableau calculé dans la routine 7500).

En outre, il faut dimensionner un tableau alphanumérique (DX$ par exemple), qui jouera le rôle de buffer d'entrées/sorties, et dont chaque élément sera associé au fichier de même numéro (voir ligne 1226). Ainsi, on posera NO=3, LN=100, NF$="A:ESSAI" et OP=1 pour ouvrir, sur le disque A, un fichier de nom ESSAI, de mémoire tampon DX$(3) et contenant des enregistrements de 100 caractères. Les opérations de lecture et d'écriture présentent une certaine complexité, et nous allons les étudier de façon plus détaillée.

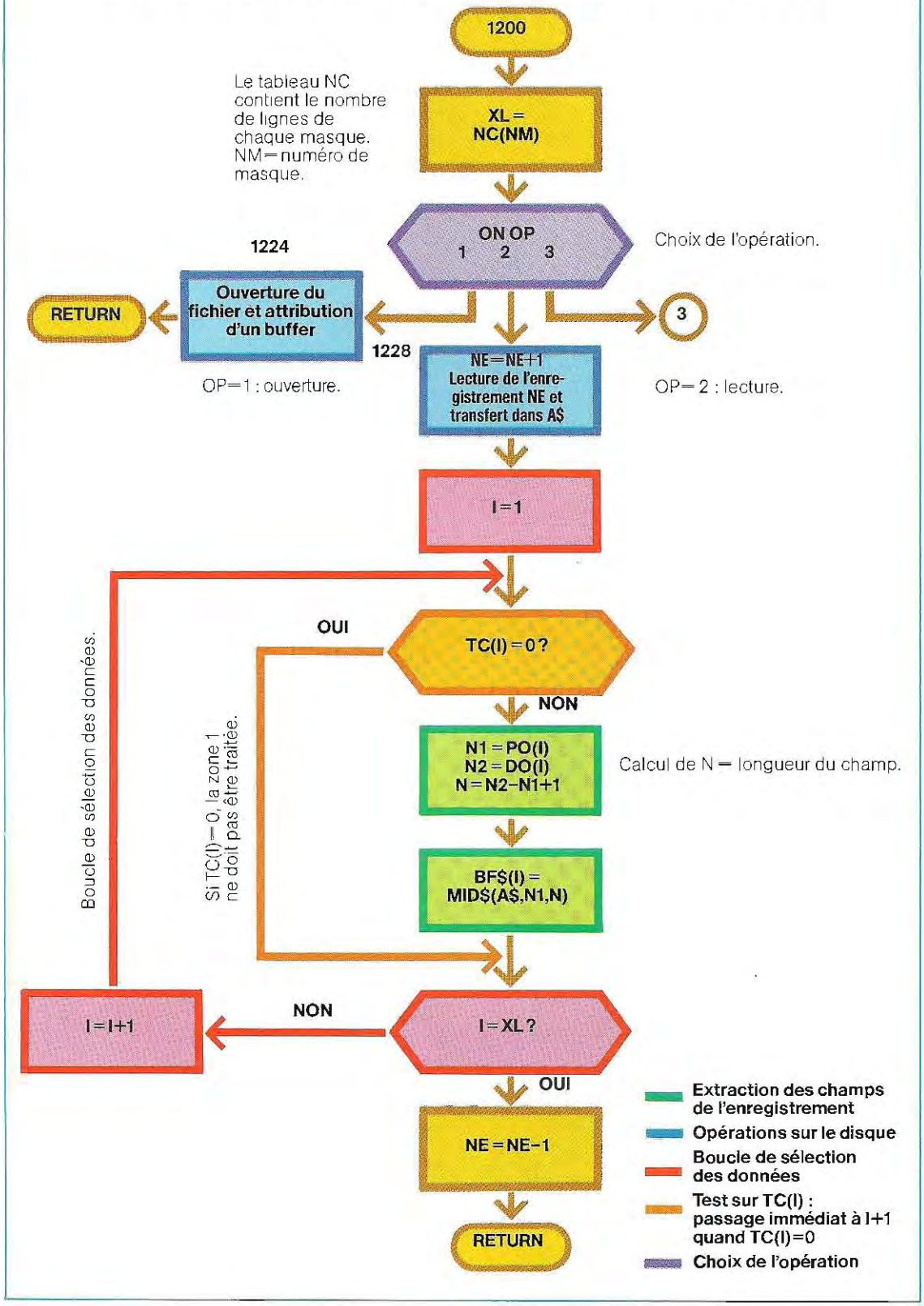
L'écriture (instructions 1242 à 1272). Conformément à la méthode que nous avons plusieurs fois mise en œuvre, le numéro du dernier enregistrement écrit est mémorisé dans le premier enregistrement, qui ne représente donc pas une donnée.
Ce décalage risquerait de provoquer des erreurs si l'utilisateur devait le contrôler lui-même, en ajoutant 1 avant de lire ou d'écrire une donnée et en s'en abstenant pour le premier enregistrement. On préférera donc le gérer automatiquement à l'intérieur même du sous-programme. Pour cela, on donnera le numéro 0 au premier enregistrement (NE=0), ce qui permettra d'ajouter systématiquement 1 pour accéder à n'importe quel enregistrement. Il faut naturellement soustraire 1 à NE avant de sortir du sous-programme, afin de lui rendre sa valeur au moment de l'appel.
Avant d'écrire un enregistrement, on doit regrouper les zones BF$ qui contiennent les données saisies à l'aide du masque.
On prépare les enregistrements de la façon suivante : après avoir cadré le contenu des buffers de saisie (BF$), on les concatène dans un ordre déterminé de manière à créer une chaîne A$, comme le montre le schéma de la page 713 et les instructions 1246 à 1263. On transfère ensuite cette chaîne A$ dans le

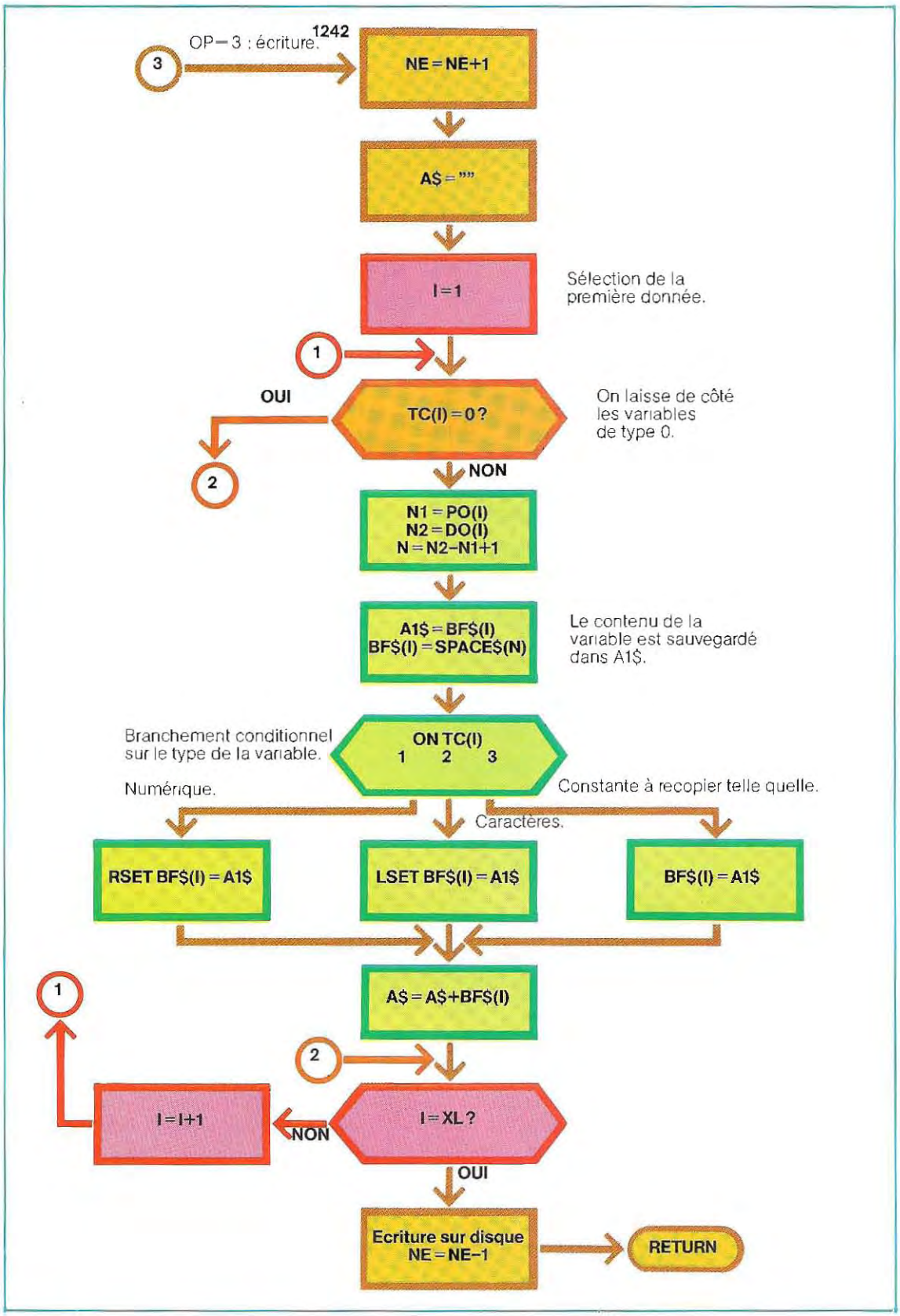
Techniciens vérifiant le fonctionnement du matériel dans un centre informatique.



K. Reese/Marka

SOUS-PROGRAMME PARAMETRE DE GESTION DU DISQUE




OP – 3 : écriture.
1242
3
NE = NE+1
A$ = ""
I = 1
Sélection de la première donnée.
1
OUI
TC(I) = 0 ?
On laisse de côté les variables de type 0.
2
NON
N1 = PO(I)
N2 = DO(I)
N = N2–N1+1
A1$ = BF$(I)
BF$(I) = SPACE$(N)
Le contenu de la variable est sauvegardé dans A1$.
Branchement conditionnel sur le type de la variable.
ON TC(I)
1 2 3
Numérique.
Constante à recopier telle quelle.
Caractères.
RSET BF$(I) = A1$
LSET BF$(I) = A1$
BF$(I) = A1$
A$ = A$+BF$(I)
1
2
I = I+1
NON
I = XL ?
OUI
Ecriture sur disque
NE = NE–1
RETURN

SOUS-PROGRAMME PARAMETRE DE GESTION DU DISQUE

```
1200 ' ** SOUS-PROGRAMME DISQUE **
1201 '    Lecture/ecriture sur disque
1202 '    FICHIER = ENCYCL
1203 '    ENTREES :
1204 ' NE = numero de l'enregistrement
1205 ' NO = numero du fichier = numero du buffer ; NF$ = nom du fichier
1206 ' OP = code operation
1207 '      1 : ouverture
1208 '      2 : lecture
1209 '      3 : ecriture
1210 ' LN,PO(*),DO(*),NC(*),TC(*)    parametres de la routine 7500
1218 '
1219 '
1220 XL=NC(NM)          ' Nombre de lignes du masque
1221 '
1222 ON OP GOTO 1224,1228,1242
1223 '
1224 ' **** OUVERTURE ****
1226 OPEN "R",NO,NF$,LN : FIELD NO,LN AS DX$(NO) : RETURN
1227 '
1228 ' **** LECTURE ****
1229 ' NE=NE+1     ' Incrementation pour obtenir le veritable numero d'enregistrement
1230 GET NO,NE : A$=DX$(NO)
1232 FOR I=1 TO XL : IF TC(I)=0 GOTO 1236
1234 N1=PO(I) : N2=DO(I) : N=N2-N1+1 : BF$(I)=MID$(A$,N1,N)      ' Extraction
1236 NEXT I
1237 NE=NE-1          ' Decrementation pour retrouver la valeur initiale
1238 RETURN
1240 '
1242 ' **** ECRITURE ****
1243 NE=NE+1
1244 A$=""
1246 FOR I=1 TO XL : IF TC(I)=0 THEN GOTO 1264
1248 N1=PO(I) : N2=DO(I) : N=N2-N1+1
1250 A1$=BF$(I)
1252 BF$(I)=SPACE$(N)
1254 ON TC(I) GOTO 1256,1258,1260
1256 RSET BF$(I)=A1$ : GOTO 1262
1258 RSET BF$(I)=A1$ : GOTO 1262
1260 BF$(I)=A1$
1262 A$=A$+BF$(I)
1263 NEXT I
1264 '
1265 PRINT "A=";A$ : S$=INPUT$(1)
1268 LSET DX$(NO)=A$
1270 PUT NO,NE
1271 NE=NE-1
1272 RETURN
```

buffer d'entrées/sorties associé au fichier (ligne 1268), et ce dernier sera recopié sur disque à la place qui lui revient (ligne 1270).

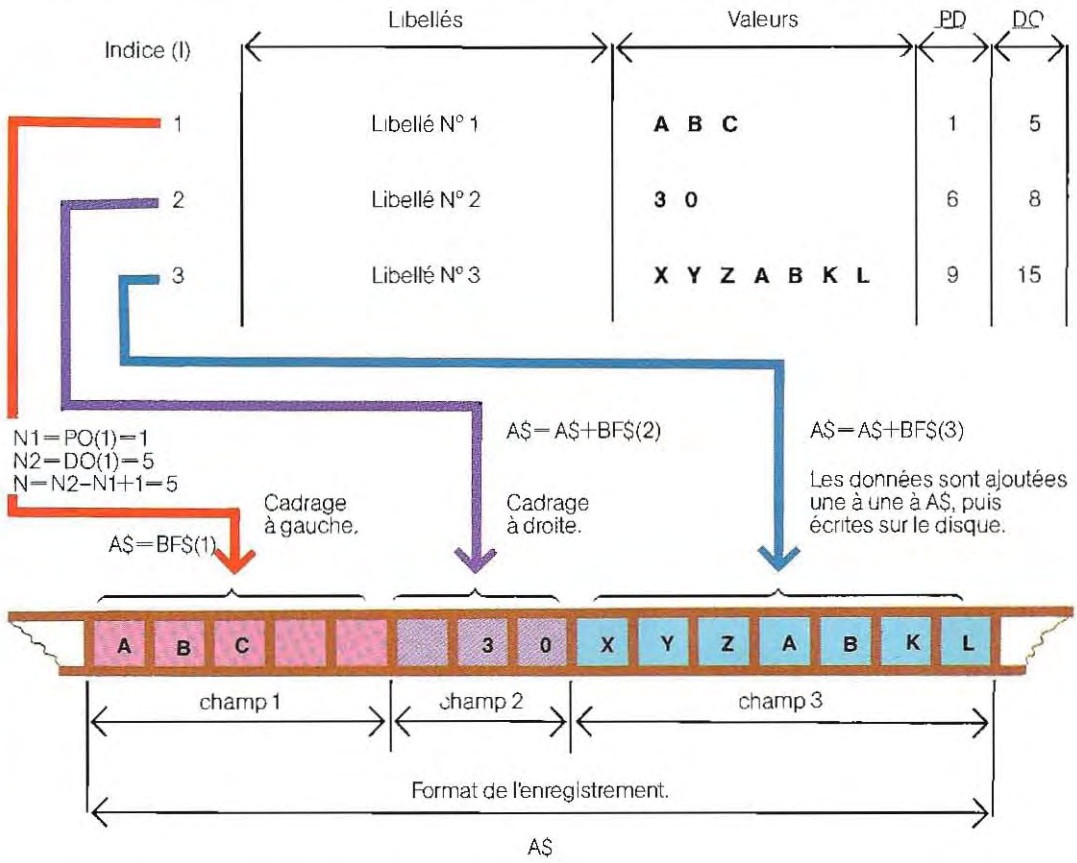
La lecture (lignes 1228 à 1238). Son mécanisme est exactement l'inverse du précédent. Après avoir extrait du disque l'enregistrement cherché (sans oublier d'ajouter 1), on transfère le contenu du buffer d'E/S du fichier dans la chaîne A$, puis on répartit les différents champs dans leurs buffers respectifs (BF$). Ce processus d'extraction est illustré par le schéma du bas de la page 713.

Le programme principal

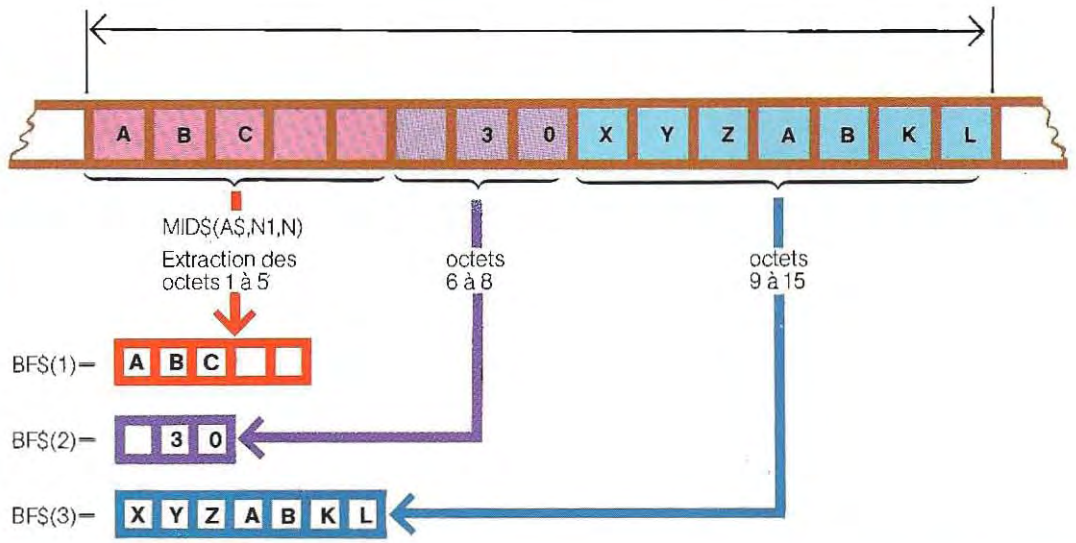
Nous avons conçu un programme principal afin de tester nos différentes routines. Son organigramme se trouve page 704 et son listing pages 715 et 716, suivi de celui du sous-programme 9000 de programmation des touches de fonction. Les codes correspondants à ces touches sont initialisés par les instructions 240 et 250 de DATA. On en a défini vingt, en prévision de développements futurs.

PREPARATION D'UN ENREGISTREMENT AVANT ECRITURE

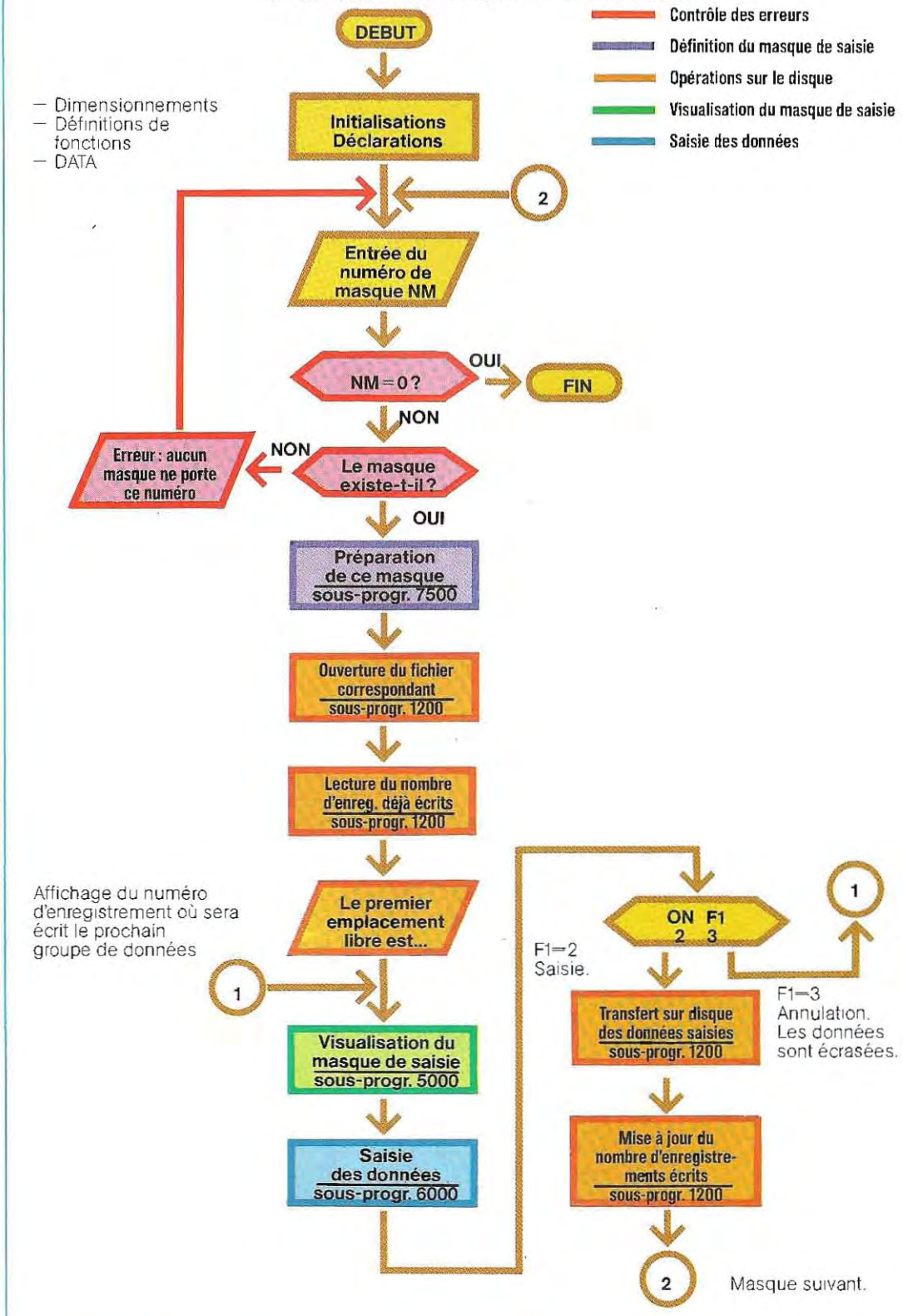
On construit un enregistrement en concaténant les différents champs par l'instruction A$=A$+BF$(I).



EXTRACTION DES CHAMPS DE DONNEES APRES LECTURE



PROGRAMME PRINCIPAL DE TEST



PROGRAMME PRINCIPAL DE TEST

```
10 ' ** PROGRAMME PRINCIPAL **
20 '    FICHIER = ENCYCL
30 OPTION BASE 1
40 '
50 ' **** DEFINITIONS ****
60 ' TF(*)  = codes des touches de fonction (8000)
70 ' DX$(*) = buffer d'E/S du fichier disque N° * (1200)
80 ' TC(*)  = type du champ * (7500)
90 ' DO(*)  = dernier octet du champ * (7500)
100 ' PO(*) = premier octet du champ * (7500)
110 ' LI$(*) = libelle du champ * (7500)
120 ' NC(*)  = nombre de champs du masque * (7500)
130 ' BF$(*) = buffers de saisie (6000)
140 ' BC$(*) = chaine de blancs pour completer les libelles (5000)
150 ' LC(*)  = longueur du champ * (5000)
160 ' BA$(*) = buffer de pointilles ou de donnee a presenter (5000)
170 '
180 '
190 '
200 '
210 DEFINT A-Z
230 DIM TF(20)          ' Tableau de 20 codes de fonctions
240 FOR I=1 TO 20 : READ TF(I) : NEXT I
250 DATA 10,8,9,11,16,17,13,0,0,0,0,0,0,0,0,0,0,0,0,0,
255 ' seules 7 touches sont utilisées. Pour en ajouter, il faut modifier le DATA
260 DIM DX$(4)
270 DIM TC(14),DO(14),PO(14),LC(14),NC(14)
280 DIM LI$(14),BF$(14),BC$(14),BA$(14),
300 '
310 '
320 '
330 ' * FONCTIONS *
340 '
350 BL$="                                        "
360 BP$="........................................"
370 BI$=CHR$(27)+"+"+CHR$
390 '
400 GOSUB 9000        ' Programmation des touches de fonction
410 PRINT BI$         ' Efface l'ecran et emet un bip
420 '
500 ' ** SELECTION DU MASQUE DE SAISIE **
510 INPUT "Taper le numero du masque desire ";NM
520 IF NM=0 THEN GOTO 900          ' Sortie
530 IF NM<1 OR NM>2 THEN PRINT "ERREUR : ce masque n'existe pas"
540 IF NM<1 OR NM>2 THEN S$=INPUT$(1) : GOTO 510
550 '
560 ' ** PREPARATION DU MASQUE **
570 GOSUB 7500          ' Chargement des DATA
575 '
580 ' ** OUVERTURE DU FICHIER **
590 OP=1 : GOSUB 1200
600 ' Lecture de l'enregistrement 1
610 NE=0 : OP=2
620 GOSUB 1200
630 DE=VAL(BF$(1))      ' Dernier enregistrement ecrit
640 NE=DE+1             ' Premier emplacement libre
650 PRINT "Le premier emplacement libre est : ";NE
660 PRINT "Appuyer sur une touche pour continuer" : S$=INPUT$(1)
665 '
670 ' ** VISUALISATION DU MASQUE DE SAISIE **
680 GOSUB 5000
685 '
690 ' ** SAISIE DES DONNEES **
700 GOSUB 6000
```

```
710 ' **BRANCHEMENT CONDITIONNEL (VALEUR DE F1)
720 K=F1-1          'On soustrait 1 : valeurs possibles 1 ou 2 au lieu de 2 et 3
725 '
730 ON K GOTO 740,870
735 '
740 '*F1=2 : SAISIE*
750 OP=3            'operation=ecriture
760 GOSUB           'Ecriture des donnees dans l'enregistrement NE
770 '
780 'Remise a jour du premier enregistrement
790 A$=STR$(NE)             'Conversion en ASCII du nombre d'enregistrements
800 NE=0                    'Le premier enregistrement est designe par 0
810 '                        On utilise le buffer BF$(1) pour transferer NE
820 N=DO(1)-PO(1)+1         'Nombre de caracteres de BF$(1)
830 BF$(1)=SPACE$(N)        'Mise a blanc de BF$(1)
840 RSET BF$(1)=A$          'Cadrage a droite du contenu de BF$(1)
850 OP=3 : GOSUB 1200       'Ecriture
860 GOTO 640                'Entree suivante
865 '
870 ' **ANNULATION DES DONNEES**
880 PRINT BI$
885 GOTO 640
890 '
900 ' **SORTIE**
910 PRINT BI$
920 PRINT "*** AU REVOIR ***"
930 END
```

```
9000 ' **PROGRAMMATION DES TOUCHES DE FONCTION**
9010 '
9020 '    FORME DU CURSEUR
9030 PRINT CHR$(27)+"c4AA"
9040 '
9050 '
9060 'Les touches de deplacement du curseur (codes 10, 8, 12 et 11)
9065 'sont gerees dans la routine 6000, ainsi que celle de justification (code 13)
9066 'On code ici 8 touches (b a i) en leur attribuant les valeurs 16 a 23.
9067 'Les deux premieres sont utilisees dans le programme
9068 'et les suivantes laissees a la disposition de l'utilisateur
9069 'qui devra bien sur les ajouter au DATA et definir leurs fonctions.
9070 A$=CHR$(27)+CHR$(48)    'Premiere etape de la programmation des
9080 '                          touches de fonction
9090 B=38                    'Code de la premiere touche de fonction (38 = code ASCII de &)
9100 I1=49                  '49 et 48 ASCII=10 HEXA car 49=1 et 48=0
9110 I2=48                  'et 10 HEXA equivaut a 16 decimal
9120 FOR I=1 TO 8           'Boucle de programmation des touches b a i
9122 'La valeur decimale 16 est associee a la premiere touche de fonction (n°98)
9130 C$=CHR$(I1)+CHR$(I2)   'Code a associer a la touche I (1 a 8)
9140 D$=CHR$(B)             'Selection de la touche
9150 E$=A$+D$+C$            'Chaine contenant les codes de commande
9160 PRINT E$               'La touche est programmee
9170 B=B+1                  'Touche suivante
9180 I2=I2+1                'Code suivant (incremente de 1)
9190 NEXT I
9200 RETURN
```

La compilation

Il est beaucoup plus commode de travailler en mode interprété pendant les phases de conception et de mise au point des programmes Basic. Par contre, l'exécution d'un programme sous le contrôle de l'Interpréteur est très longue pour trois raisons fondamentales :

- à chaque instruction d'appel (GOTO ou GOSUB), l'Interpréteur doit lire entièrement le programme jusqu'à la ligne où il doit poursuivre l'exécution ;
- quand un nom de variable est mentionné, l'Intepréteur doit chercher dans la table des variables l'emplacement de mémoire où se trouve sa valeur ;
- l'Interpréteur doit traduire chaque instruction en langage machine avant de l'exécuter. En particulier, dans une boucle, il décodera les instructions Basic à chaque itération. Or, si la traduction d'une instruction s'effectue très rapidement, on finit par atteindre des temps d'exécution assez longs quand on exécute de nombreux passages.

Tous ces inconvénients sont liés à la manière dont fonctionne l'Interpréteur.
On les évitera en **compilant** les programmes après leur mise au point. Le Compilateur se charge de traduire les instructions en binaire et de convertir les noms de variables en adresses absolues. L'exécution d'un programme compilé est donc bien plus rapide (4 à 10 fois). De plus, le Compilateur effectue un diagnostic global et relève éventuellement des erreurs qui ont pu échapper à l'Interpréteur.
C'est notamment le cas pour les instructions de branchement conditionnel. Si la ligne à laquelle renvoie l'instruction de saut n'existe pas, l'Interpréteur ne s'en apercevra (et, donc, ne le signalera) qu'au moment d'effectuer le saut, c'est-à-dire seulement si la condition est vérifiée.
L'erreur pourra donc passer inaperçue pendant la mise au point, et ne se révéler qu'après la mise en service du programme, ce qui risquera de fausser les données en cours de traitement, ou même d'entraîner leur perte.
En revanche, la compilation inclut un contrôle systématique des adresses, qui permet de déceler les erreurs éventuelles de ce type.

Le schéma de la page 718 retrace la succession des opérations d'écriture, de compilation et d'exécution d'un programme Basic.
Nous allons maintenant décrire en détail ces différentes étapes.

La préparation

Le Compilateur ne peut opérer que sur des programmes sauvegardés en ASCII. Il faut donc, avant tout, s'assurer que le programme source a bien été mémorisé sous cette forme : il doit être sauvegardé sur disquette (depuis la mémoire centrale) avec l'option "A". Ainsi, pour un programme baptisé ESSAI, on procédera de la façon suivante :

```
LOAD "A:ESSAI"
SAVE "A:ESSAI",A
```

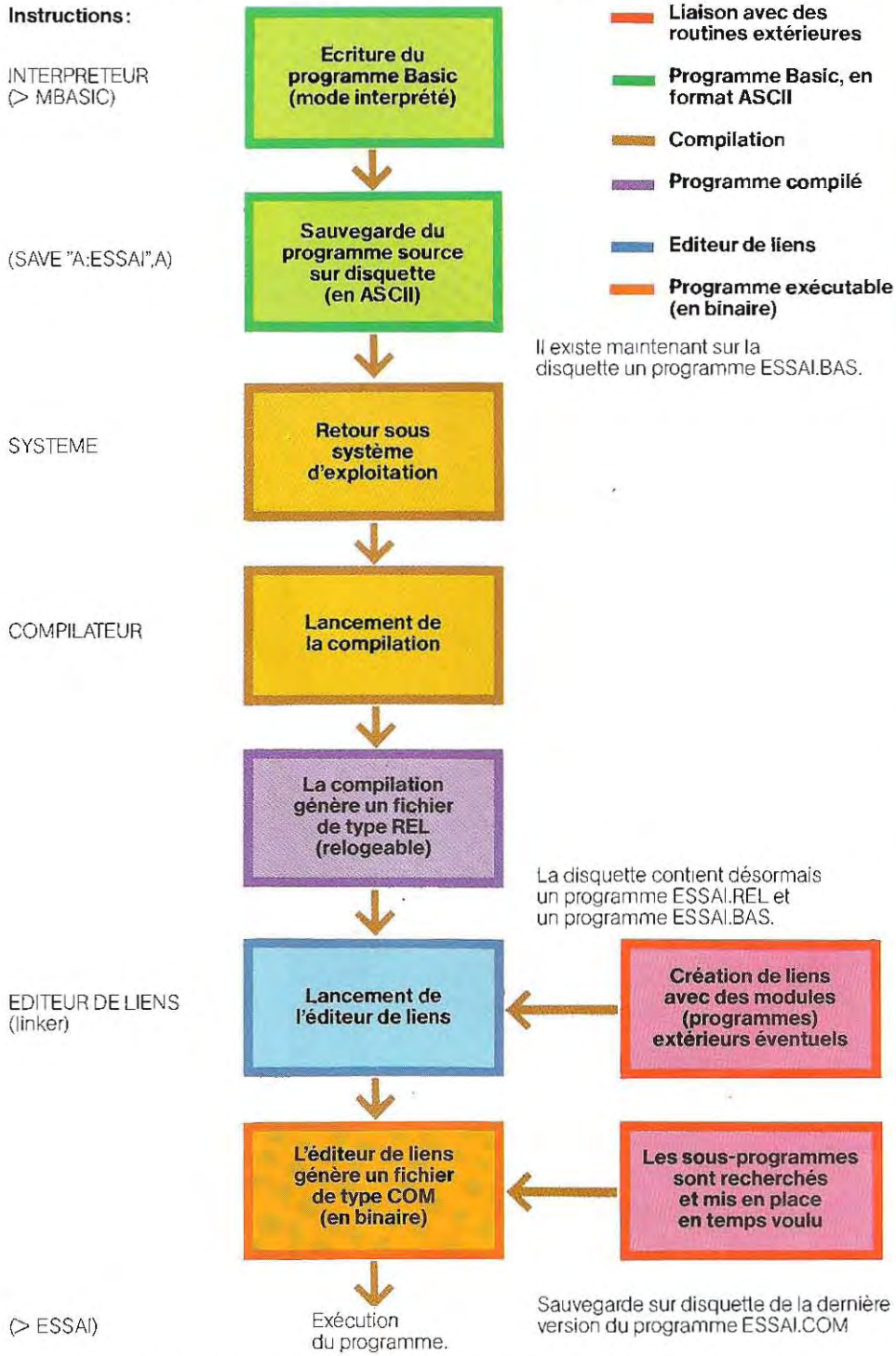
En outre, il est préférable – toujours pour raccourcir le temps d'exécution – de vérifier que les indices de toutes les boucles sont définis comme entiers, grâce à l'instruction DEFINT ou en spécifiant l'attribut % (écrire DEFINT I revient à faire suivre I du symbole %).
Enfin, il est nécessaire que tous les programmes (utilitaires du système ou non) qui seront appelés pendant la compilation, puis pendant l'« édition de liens »* (linkage) se trouvent sur la disquette A. C'est, en effet, sur l'unité A que travaille le Compilateur ; s'il est possible d'effectuer certaines opérations sur l'autre unité, il vaut mieux que tout se passe sur la même disquette. De toute façon, rien n'empêche de transférer le programme après sa compilation.
Les programmes indispensables à la compilation sont énumérés ci-dessous.

BASCOM.COM. C'est un programme système qui traduit le Basic en Assembleur. Il fournit une version compilée du programme source.

BRUN.COM. Il contient la plupart des routines utilitaires appelées par le programme d'application. Utilisé pendant l'exécution de celui-ci, il doit être présent dès la première étape afin que les liens nécessaires puissent être mis en place.

* Nous expliquerons un peu plus loin en quoi consiste l'édition de liens entre des programmes ou des parties de programmes. Considérons pour l'instant que ce terme désigne une série d'opérations qui relient des modules compilés séparément.

AVANT, PENDANT ET APRES LA COMPILATION



La disquette contient donc au moins trois versions du programme :
ESSAI.BAS = Programme source en Basic ;
ESSAI.REL = Programme compilé ;
ESSAI.COM = Programme exécutable en binaire.

Sammie et l'automobile

La conception d'une automobile doit réaliser le compromis le plus satisfaisant entre des contraintes d'ordre technico-économiques souvent antagonistes. L'équilibre est très difficile à trouver, car il faut respecter des normes de plus en plus nombreuses, répondre aux exigences d'un marché en constante évolution, tout en assurant aux véhicules d'excellentes performances.

La complexité du travail de conception ne se pose plus en termes de technologie, mais plutôt d'une multiplicité d'éléments à concilier pour offrir au consommateur un éventail de services de plus en plus diversifiés.

En effet, le véhicule familial ne représente plus aujourd'hui un simple moyen de transport utilitaire. Il doit répondre à une foule de besoins plus ou moins subjectifs.

Une bonne voiture aura une charge utile élevée et un habitacle spacieux. De ligne séduisante, elle consommera peu et sera facile à conduire pour les automobilistes peu expérimentés. Sa production, son entretien et sa réparation seront simples et peu coûteux.

Sur le plan commercial, ses ventes devront assurer rapidement les bénéfices nécessaires au financement de nouveaux modèles.

Beaucoup de temps s'écoule entre la conception d'un nouveau modèle et sa commercialisation, et il n'est pas rare que les ingénieurs et les responsables du marketing doivent imaginer les caractéristiques techniques de produits qui ne seront pas mis sur le marché avant une dizaine d'années. Ces prévisions sont suivies d'un ensemble d'études complexes et interdépendantes, depuis la réalisation de maquettes, de modèles, de prototypes, de moules et de machines-outils, jusqu'à la mise en place des équipements de production. Ce n'est qu'au terme de ce long processus que la production en série peut commencer.

On comprend donc l'intérêt des ***systèmes de CAO*** *(Conception Assistée par Ordinateur), qui réduisent considérablement les délais de conception et facilitent le recours aux techniques d'analyse structurelle, dynamique et vibratoire, qui permettent d'améliorer le produit et, par conséquent, de donner plus grande satisfaction à l'utilisateur.*

Avec les systèmes de CAO, on peut notamment rejeter tout de suite de mauvaises solutions, dont les défauts n'apparaissaient autrefois qu'après la construction des prototypes, alors qu'on manquait de temps ou d'argent pour résoudre les ultimes problèmes découverts. Les dernières étapes de la conception et les systèmes d'expérimentation non informatisés absorbaient d'énormes investissements.

De nombreux logiciels disponibles aujourd'hui rendent possible la conception de ***maquettes virtuelles tridimensionnelles*** *(3D). Ils permettent de définir simplement des formes géométriques à l'aide d'un langage évolué.*

On peut combiner à volonté ces éléments géométriques dans l'espace, pour représenter les objets les plus complexes, qu'il s'agisse d'un bâtiment, d'une pièce mécanique de moteur, d'une chaîne de production, du fuselage d'un avion, de la carcasse d'une automobile ou de n'importe quelle autre structure. Citons pour mémoire les systèmes français EUCLID (Matra-Datavision), CATIA (Dassault-Systèmes), UNISURF (Renault-Automatismes) et SISTRID (SNIAS).

Le logiciel de CAO américain SAMMIE offre la particularité intéressante de pouvoir représenter un personnage humain, et de simuler ses déplacements, facilitant ainsi toutes les études d'ergonomie en vogue actuellement. Nous avons choisi l'exemple concret de la conception d'un nouveau modèle de voiture pour mieux illustrer cette possibilité.

Le dessinateur commence par résumer les études préliminaires dans un croquis où figurent les mesures les plus significatives (longueur totale, largeur, hauteur) et les formes proposées pour chaque surface.

A partir de ces quelques données disponibles immédiatement, on peut matérialiser l'idée initiale par un modèle à trois dimensions. On se contente, dans un premier temps, de représenter les surfaces courbes de la carosserie par un assemblage de carreaux plans. Cette opération, relativement rapide, demande environ 30 % de la durée totale de la phase de conception. Ainsi, même s'il ne dispose encore que de données extrêmement sommaires, le concepteur est déjà en mesure de se faire une idée relativement précises du nouveau véhicule.

On fait ensuite « tourner » la maquette sur elle-même, afin de pouvoir l'examiner en perspective sous tous ses angles ; cette étude, avec les techniques traditionnelles, était un luxe exigeant des heures de travail sur la planche à dessin.

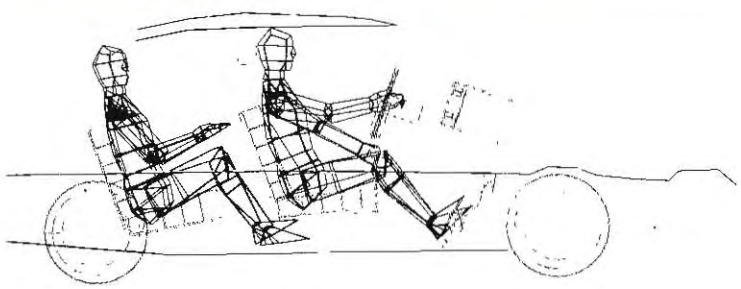
Après avoir défini la carosserie du véhicule, on commence à s'occuper de l'habitacle. Le concepteur doit alors tenter de réutiliser le plus grand nombre possible d'éléments ayant déjà servi sur les précédents modèles. On pourra, dans certains cas, reprendre les sièges d'une voiture antérieure et les adapter aux dimensions de la nouvelle carroserie. Il suffira, pour cela, de rechercher dans la ***base de données*** *de SAMMIE les maquettes créées auparavant. On réduit non seulement la durée mais le coût des études.*

Il n'est évidemment pas possible d'appliquer cette méthode à tous les éléments de l'habitacle. La disposition ergonomique des appareils du tableau de bord soulève des difficultés particulières, car le conducteur doit pouvoir les consulter, à tout moment, d'un seul coup d'œil et sans effort.

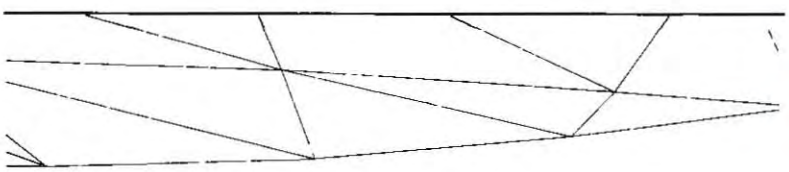
Il faut donc les placer à la fois le plus haut et le plus loin possible du conducteur. Cette nécessité est, bien sûr, en contradiction avec le besoin de visibilité et les contraintes de la géométrie extérieure du véhicule.

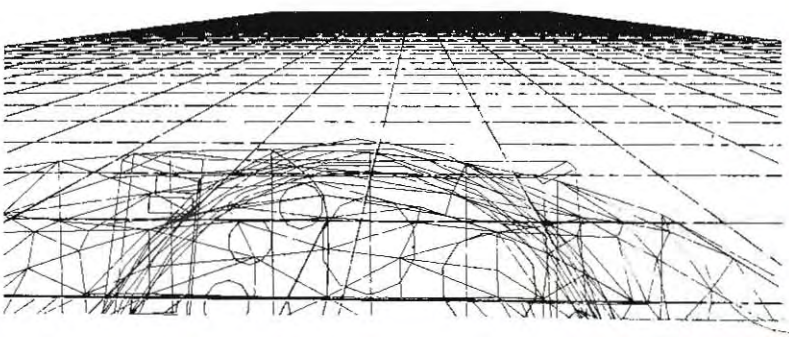
On peut résoudre ce genre de problèmes en recourant aux méthodes classiques mais la CAO offre une toute autre efficacité.

SAMMIE autorise une véritable ***« modélisation sous contraintes »,*** *qui prend en compte des données comme l'occupation de l'espace, les intersections et tangences autorisées, etc.*

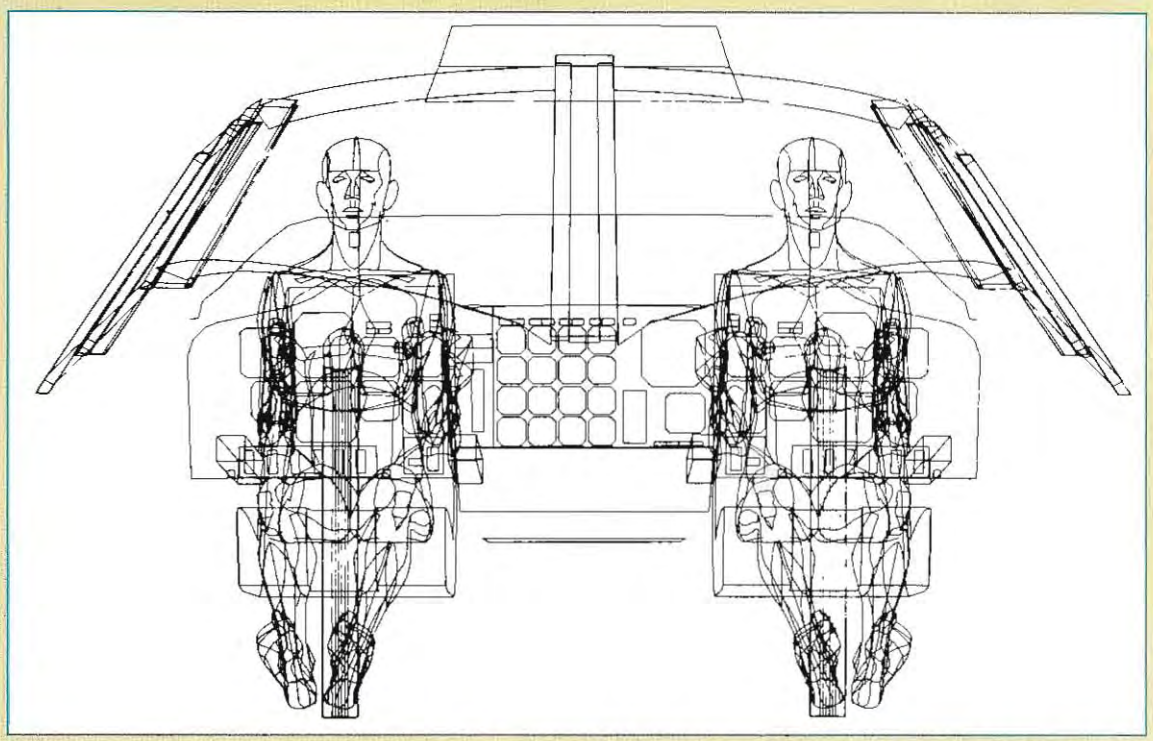


Les logiciels de CAO permettent aujourd'hui la conception rationnelle et rapide des structures les plus complexes.





Ces images ont été réalisées par une table traçante contrôlée par le logiciel SAMMIE. Elles illustrent deux étapes de la conception de l'habitacle d'un véhicule de tourisme. L'ingénieur peut s'assurer que les sièges, le volant et le tableau de bord se placent correctement les uns par rapport aux autres (en haut) et que la visibilité est satisfaisante (ci-contre). On aperçoit le tableau de bord au premier plan.



IBM

Deux mannequins assis dans la cabine de pilotage d'un Boeing. On notera la précision de la position du corps.

En pratique, le concepteur effectue une douzaine de tentatives qui, avec un système manuel, auraient duré plusieurs semaines. L'aménagement de l'habitacle prend environ 25 % du temps de conception.

SAMMIE peut représenter un personnage humain, opérateur ou homme-type, et le situer dans son milieu de travail. Le mannequin reproduit les caractéristiques cinématiques du corps humain : une suite de joints d'articulation, reliés à des membres rigides, schématise l'allure et la silhouette d'un personnage réel.

A première vue, on pourrait peut-être penser qu'une modélisation aussi rigide est inutile, et que le même résultat s'obtiendrait par des équations mathématiques. Mais on perdrait alors cette ***communication instantanée*** *par le biais de l'image. Le mannequin donne en fait une idée immédiate de l'orientation et de la position prise par le corps humain, et le concepteur peut ainsi vérifier facilement les relations qui s'établissent entre l'homme et son milieu de travail.*

La base de données utilisée par Sammie pour la création de l'homme-type peut en outre se modifier facilement, afin d'analyser la diversité des comportements due à la variété anatomique présente dans le monde réel.

Le mannequin est ensuite articulé dans tous ses joints, mais ses mouvements sont contrôlés par une table qui l'empêche de prendre des positions anormales. Le concepteur peut donc intervenir sur cette table pour simuler, par exemple, l'effet d'une contrainte quelconque, anatomique ou imposée par le milieu.

A la flexibilité du modèle, correspond la souplesse d'emploi des commandes qui feront bouger les articulations. Lorsqu'une commande est émise, le système répond à différents niveaux, par des rotations ***explicites*** *(mouvement demandé) et* ***implicites*** *(mouvement induits par les premiers) ; la précision de la réponse dépend du type particulier de mannequin que l'on a choisi.*

En retournant maintenant à notre maquette d'automobile, constituée de surfaces externes et d'objets internes, nous remarquons que le concepteur peut réellement se mettre à la place du conducteur. Il lui suffira pour cela de faire coïncider le champ de vision du mannequin (qui normalement doit regarder devant lui), avec l'angle sous lequel il veut voir le modèle en perspective.

Il pourra ainsi jeter un œil au tableau de bord, se faire une idée exacte de l'amplitude du déplacement que l'œil doit effectuer pour cette opération, et déterminer la partie de route qui sera alors exclue de son champ de vision. Afin de rendre cette vérification encore plus réaliste, on peut disposer des cylindres sur le tableau de bord, de façon à simuler la présence d'instruments.

Grâce au module SIGHT (repère) et un modèle de volant, on peut ensuite étudier différentes alternatives, afin de minimiser l'encombrement du tableau de bord et la réduction du champ de vision. Des améliorations ultérieures seront pratiquées, par une étude paramétrée de ces effets en fonction de la taille du conducteur.

Après avoir résolu le problème de la visibilité, il faut vérifier les possibilités de manœuvre. Le module REACH (atteindre) permet au concepteur de demander au mannequin d'essayer de toucher un point quelconque autour de lui, avec les doigts, la paume d'une main, le talon ou la semelle de sa chaussure. Ce module fonctionne avec un positionnement approximatif obtenu grâce au curseur, ou bien avec une précision géométrique fondée sur les coordonnées tridimensionnelles. En pratique, il permet de vérifier si le conducteur peut atteindre les instruments et si les manettes de commandes sont placées correctement les unes par rapport aux autres.

Grâce à la nature **hiérarchisée** du système, les fonctions cinématiques peuvent s'appliquer également aux composants mécaniques. Par exemple, les pédales et le levier de vitesses peuvent entrer en rotation autour d'un système de coordonnées locales, afin de simuler les positions limites et de vérifier l'accessibilité dans tous les cas, par une simple itération du calcul. On évitera ainsi, par exemple, qu'en quatrième, le levier de vitesses donne des crampes à 95 % des conducteurs, ou que 5 % d'entre eux n'aient à accomplir un effort excessif pour engager la première ou la marche arrière. Cette possibilité de modification rapide des perspectives, joint à une vaste bibliothèque et à l'**interactivité** entre la maquette et l'homme-type, permet une réduction notable du temps de conception, par rapport aux méthodes traditionnelles de type manuel.

Après avoir modélisé l'intérieur, il faut revenir plus attentivement aux questions d'esthétique et d'aérodynamisme qui, dans notre cas, impliquent des modifications de la carrosserie du véhicule. Ces **surfaces curvilignes** avaient été schématisées dans un premier temps, au moyen d'interpolations triangulées. La maquette extérieure doit maintenant être affinée, et prendre en compte de complexes descriptions de surfaces, à partir de modèles mathématiques très précis. Une fois définies, ces surfaces courbes sont intégrées à la structure hiérarchique de la maquette. Disposant d'un modèle mathématique rigoureux, le concepteur peut effectuer tous les calculs classiques d'aérodynamisme, de résistance des matériaux... comme s'il disposait d'un prototype classique.

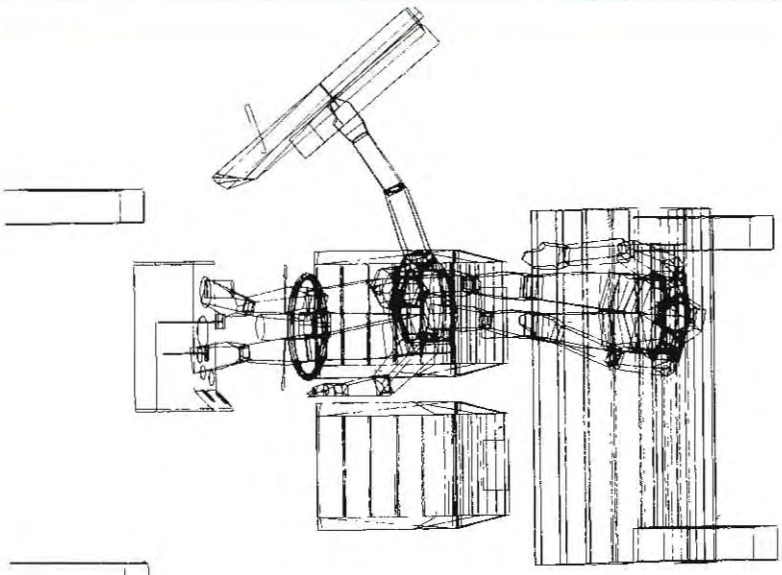
Après ce **lissage** de la surface externe de la carrosserie, on peut réévaluer la visibilité des instruments et de la route. Par exemple, on ajoutera un quadrillage représentant le terrain, et on y placera des modèles d'objets (des piliers, des feux de signalisation, etc.) pour mesurer la réduction de visibilité due à la présence d'un passager sur le siège avant, ou au rétroviseur. On peut ensuite dessiner une carte complète de la visibilité et obtenir des informations très utiles, avant même de disposer de maquettes de bois ou de prototypes.

SAMMIE propose également un module appelé MIRROR (miroir), qui transforme certaines surfaces internes du modèle en miroirs, pourvus des caractéristiques optiques nécessaires pour renvoyer en perspective les images réfléchies.

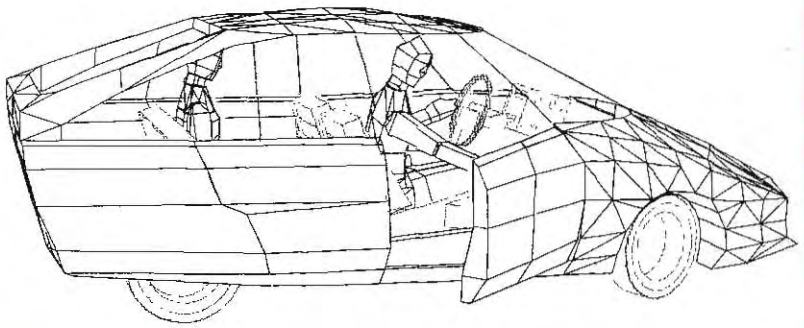
L'utilisateur peut aussi se servir de la commande ADJUST FOR OBJECT (vise un objet), qui, se fondant sur la dernière perspective appelée, modifie la position du miroir de façon que la bissectrice des deux vecteurs (allant de l'œil du conducteur au miroir et du miroir à l'objet à viser) soit perpendiculaire à la surface du miroir. Par cette opération, l'image de l'objet en question apparaît en perspective dans le miroir, avec des zones d'ombre ou d'interférences éventuelles.

L'élimination des parties cachées constitue la pierre d'achoppement des petits systèmes de CAO, pour la quantité de temps machine qu'elle demande.

SAMMIE offre la possibilité d'éliminer les parties cachées, mais son mode d'utilisation normale se limite à construire un modèle



L'intégration de commandes appropriées permet de contrôler les possibilités de mouvement et de manœuvre des personnages assis dans l'habitacle. Ici, Sammie vérifie que la poignée de la portière est à portée de main.



Un sérieux problème, qui apparaît dans la conception assistée par ordinateur, est le manque de lisibilité des images reproduites, dû à la présence de lignes internes qui, dans la réalité, seraient masquées par les surfaces externes. Avec Sammie, on peut les éliminer par une simple commande.

*représenté par un ensemble de lignes (**structure filaire** ou « fil de fer »), à le faire tourner et à mémoriser les différentes perspectives. A la fin de sa journée, le concepteur peut « mettre au propre » les perspectives qu'il a réalisées grâce à une fonction de HIDDEN LINES et les conserver comme documentation.*

La conception d'un projet suit donc le même cheminement qu'avec les techniques traditionnelles, mais se réduit à des temps beaucoup plus courts, et le système fournit toutes les informations nécessaires pour construire une maquette satisfaisante du premier coup. Toutes les évaluations dont nous avons parlé s'effectuent avec une précision et une vitesse supérieure à celle qu'on obtiendrait avec un crayon, du papier et une machine à dessiner. Par ce moyen, le concepteur se libère des tâches répétitives et ennuyeuses, pour se concentrer sur l'aspect le plus créatif de son travail.

Actuellement, les systèmes de CAO permettent de réduire les coûts des projets dans un rapport variant de 5 à 1 jusqu'à 100 à 1 et davantage. Généralement, ces évaluations n'intègrent pas le coût de réalisation de la maquette réelle, mais même en en tenant compte, le bilan demeure tout de même très positif.

Les systèmes de CAO actuels, extrêmement puissants, sont implantés sur de grosses machines (IBM, VAX...) et accessibles seulement à quelques entreprises très importantes. A l'opposé, seuls des logiciels graphiques plus ou moins rudimentaires (2D et non volumique) peuvent tourner sur micro-ordinateur. Il est regrettable que rien ne soit proposé entre ces deux extrêmes, et l'avenir appartient peut-être aux systèmes de CAO tournant sur micro-ordinateur, destinés aux PMI et PME, aux bureaux de stylisme ou aux cabinets d'architectes.

BASLIB.REL. Contient d'autres programmes qui complètent la bibliothèque BRUN.COM et qui peuvent être appelés par les programmes d'application. Le programmeur accède aux programmes système (par exemple, la fonction SQR) sans spécifier où ils résident. Tout le travail de traduction et d'édition de liens est pris en charge par le système, au moyen du Compilateur et du Linker (plus précisément Linking Loader). Il analyse chaque ligne du programme source, reconnaît l'appel à un programme système, recherche ce dernier parmi les différentes bibliothèques du système d'exploitation et établit les adressages corrects qui en permettront l'appel pendant l'exécution du programme d'application.

OBSLIB.REL. Contient des programmes et fonctions similaires à ceux de BRUN et BASLIB. La distinction réside dans un mode de gestion différent, conséquence d'une utilisation différente du programme d'application.
Pour certaines applications, le logiciel une fois mis au point ne sera plus jamais modifié, et l'utilisateur voudra l'implanter définitivement sur la machine. Dans ce cas, il sera stocké sur une mémoire ROM (accessible en lecture seulement) ; on le qualifie alors de « software permanent » (firmware). De cette manière, on obtient l'activation automatique du programme dès la mise sous tension de la machine ; l'utilisateur final n'a pas à se préoccuper de la phase de chargement. On adopte cette solution dans des cas particuliers qui répondent à deux critères fondamentaux :

- La machine et le programme constituent ensemble un système figé, consacré à une application précise immuable (sinon par le remplacement des mémoires).
- Ce système doit être produit en série, car il ne serait pas intéressant de préparer une mémoire ROM pour une application ponctuelle.

L'écriture d'un programme sur une mémoire ROM implique un mode de travail différent, puisqu'il n'est pas possible d'écrire des valeurs dans l'espace mémoire réservé à celui-ci. Les variables seront stockées dans une mémoire RAM, séparée de la mémoire ROM qui contient le programme, ce qui nécessitera des procédés plus complexes que d'habitude. Dans la pratique, cette structuration contraint à utiliser deux espaces mémoires distincts : l'espace programme (mémoire ROM) et l'espace données (mémoire RAM). Ces options particulières sont spécifiées à l'Editeur de liens, à l'aide d'indicateurs spéciaux qui demandent le chargement de la bibliothèque OBSLIB à la place des bibliothèques BRUN et BASLIB.

BCLOAD. Ce fichier contient les liens, c'est-à-dire les références aux emplacements mémoire où se trouvent les différentes parties du programme, ainsi que les programmes système appelés par ce dernier.

L80.COM. Ce programme, exécuté après BASCOM, établit les liens nécessaires au chaînage des différentes parties, et génère ainsi un programme final exécutable. Ce module Linking Loader (Editeur de liens) tient son nom des fonctions même qu'il exécute. Certains systèmes offrent d'autres fonctions spécifiques dont certaines seront décrites par la suite.

La compilation

Le compilateur (BASCOM) lit en entrée un fichier source (programme en Basic), et produit en sortie un fichier compilé (programme traduit en Assembleur), avec toutes les références à des adresses relatives.
Lors de l'écriture d'un programme, pour affecter la valeur 2.5 à une variable V, on utilisera l'instruction V=2.5, qui code sous une forme simple et synthétique l'opération à effectuer. Pour la machine, cette instruction n'a pas de sens ; il est nécessaire de la décoder, c'est-à-dire de spécifier l'adresse mémoire attribuée à la variable V et de traduire l'instruction en langage machine (transfert de la valeur 2.5 à l'adresse de V).
Pendant la compilation, toutes les instructions du programme sont traduites à tour de rôle, et à chaque variable se voit attribuer une adresse d'emplacement mémoire. Toutes les adresses utilisées sont **relatives** à une valeur d'index, déterminée par le système : en changeant cette unique valeur, on modifie l'implantation en mémoire de toutes les variables. Ainsi, le programme compilé est dit **relogeable**.
Le calcul qui, à partir des valeurs relatives,

permet d'obtenir les adresses absolues (physiques) dans la mémoire, est exécuté par l'**éditeur de liens** (Linking Loader). Le programme compilé relogeable est mémorisé dans un fichier du type .REL. La synthaxe de la commande de compilation est

BASCOM Objet, Liste = Source.

Voici la signification de ses différentes paramètres :

Objet. Il s'agit du nom donné au fichier de sortie (de type .REL) dans lequel sera mémorisée la version relogeable du programme. Ce type de fichier est appelé « object » (Objet).

Liste. Il s'agit du fichier « liste » qui contient le listing du programme source et sa traduction. Les pages 726 et 727 représentent le contenu de ce fichier dans un exemple de compilation. Le listing contient l'expansion de chaque instruction Basic en instructions Assembleur correspondantes et retrace la compilation. Comme on peut le voir, une instruction Basic génère une série d'instructions Assembleur, contenant aussi des appels aux programmes systèmes, par exemple CALL $CINA (l'appel à un programme Assembleur utilise le code CALL et non GOSUB).
Dans le listing, chaque instruction du fichier source est précédée de deux nombres hexadécimaux. Le premier représente l'adresse de la ligne relative à l'adresse de début du programme (nulle par définition), le second représente l'espace mémoire réservé aux variables de l'instruction. Par exemple, la traduction de la ligne Basic 50 se trouve à l'adresse relative 1F (31 en décimal). Lorsque le programme est chargé en mémoire, son adresse initiale ne vaut plus zéro, mais perd une certaine valeur (origine) déterminée par le système ; ainsi l'instruction 50, avec 1F pour adresse relative, sera positionnée à une adresse absolue égale à 1F plus l'origine (adresse à laquelle le programme a été chargé).

Source. Il s'agit du fichier ASCII où réside le programme à compiler.

Appareillages d'un centre d'instruction aéronavale. Les consoles reproduisent les situations des centres opérationnels de combat.

B.A. Liotta/Il Dagherrotipo

EXEMPLE DE COMPILATION

```
BASCOM 5.30 - Copyright 1979,80,81 (C) by MICROSOFT - 25298 Bytes Free
 0014 0007       10 ' ** ESSAI DE COMPILATION **
 0014 0007       20 OPTION BASE 1
 0014 0007       30 DIM A(5)
 0014 0007       40 FOR I=1 TO 5
        ** 0014' I00000: CALL    : $530
        ** 0017' L00010: L00020: L00030: L00040:
        ** 0017'         CALL    $LFMA
        ** 001A'         DW      <const>
        ** 001C'         JMP     I00001
        ** 001F' I00002:
 001F 001B       50 B=I*A(I)
        ** 001F' L00050: LXI     H,I!
        ** 0022'         CALL    $CINA
        ** 0025'         DAD     H
        ** 0026'         DAD     H
        ** 0027'         LXI     D,A!+FFFC
        ** 002A'         DAD     D
        ** 002B'         CALL    $SLHA
        ** 002E'         CALL    $FMUC
        ** 0031'         DW      I!
        ** 0033'         CALL    $FASD
        ** 0036'         DW      B!
 J038 0023       60 NEXT I
        ** 003B' L00060: CALL    $FADA
        ** 003B'         DW      I!
        ** 003D'         DW      <const>
        ** 003F' I00001:
        ** 003F'         CALL    $FASD
        ** 0042'         DW      I!
        ** 0044'         CALL    $LEJA
        ** 0047'         DW      I!
        ** 0049'         DW      <const>
        ** 004B'         DW      I00002
 004D 0023       70 FOR I=1 TO 5
        ** 004D' L00070: CALL    $LFMA
        ** 0050'         DW      <const>
        ** 0052'         JMP     I00003
        ** 0055' I00004:
 0055 0023       80 PRINT "I = ";I," A(I) = ";A(I)
        ** 0055' L000080: CALL   $PROA
        ** 0058'         LXI     H,<const>
        ** 005B'         CALL    $PVID
        ** 005E'         LXI     H,I!
        ** 0061'         CALL    $PVOA
        ** 0064'         LXI     H,<const>
        ** 0067'         CALL    $PVID
        ** 006A'         LXI     H,I!
        ** 006D'         CALL    $CINA
        ** 0070'         DAD     H
        ** 0071'         DAD     H
        ** 0072'         LXI     D,A!+FFFC
        ** 0075'         DAD     D
        ** 0076'         LXI     $LFHA
        ** 0079'         LXI     H,$AC%
        ** 007C'         CALL    $PV2A
 007F 0023       90 NEXT I
        ** 007F' L00090: CALL    $FADA
        ** 0082'         DW      I!
        ** 0084'         DW      <const>
        ** 0086' I00003:
        ** 0086'         CALL    $FASD
        ** 0089'         DW      I!
```

```
        **008B'          CALL    $LEJA
        **008E'          DW      I!
        **0090'          DW      <const>
        **0092'          DW      I00004
 0094 0023       100 END
        **0094' L00100: CALL    $END
 0097 0023
        **0097'          CALL    $END
 00B6 002D

 00000 Fatal Error(s)
 24892 Bytes Free
```

Pendant la compilation, le fichier objet est toujours généré, même si son nom n'est pas déclaré expressément dans l'appel du Compilateur. En ce cas, il porte le même nom que le fichier source. Par exemple, la commande

BASCOM = ESSAI1

déclenche la compilation et génère le fichier ESSAI1.REL, nom par lequel il devra être appelé dans la phase d'édition de liens.
Le fichier contenant le listing, par contre, n'est pas généré automatiquement ; en outre, on peut l'obtenir directement à l'écran ou sur imprimante sans demander la création d'un fichier. Dans ce cas, l'appel sera ainsi formulé :

BASCOM ESSAI2, LST: =ESSAI

La machine crée alors le fichier relogeable ESSAI2.REL et imprime le listing de la compilation au lieu d'en faire un fichier. Le code LST: est interprété comme une demande d'impression (sous CP/M, l'unité LST est l'imprimante) ; le code TTY: dirigera le listing sur l'écran. Les fichiers de provenance (Source) ou ceux de destination (Relogeable, Liste) peuvent résider sur des unités disques distinctes spécifiées au moment de l'appel. Par exemple, BASCOM B:ESSAI2, LST: =ESSAI exécute les mêmes fonctions que la commande précédente, en transférant toutefois la sortie (ESSAI2.REL) vers l'unité de disque B.
Le compilateur, tout en traduisant les instructions, détecte les erreurs de syntaxe que pourrait recéler le fichier source. Il s'avère parfois intéressant de compiler sans générer de fichier objet. En ce cas, la commande devient :

BASCOM,=Source

On obtiendra uniquement, en sortie, l'affichage de la liste des erreurs et l'espace mémoire occupé par le programme.
Pour finir, la compilation peut s'exécuter en mode « pas-à-pas » (instruction par instruction). La commande

BASCOM,TTY: =TTY:

n'utilise aucun fichier (même pas la source) et accepte les instructions à l'écran. Chaque ligne est traduite au moment de la saisie et présentée à l'écran en même temps que les diagnostics éventuels. Cette option sert, en règle générale, à la vérification formelle (syntaxique) des instructions ; elle peut s'avérer très utile à l'apprenti-programmeur, puisqu'elle jouera un rôle de guide, en signalant les erreurs, instruction par instruction.

Les fonctions particulières du Compilateur

La commande qui appelle la compilation peut se compléter par certains indicateurs (switches), qui activent à leur tour des fonctions particulières. Le même Compilateur travaillera sur différents types de Basic de manière différente ; ces indicateurs spécifient le type de Basic et la manière de traiter certaines informations particulières. Les switches se divisent en trois catégories :

- spécification de conventions ;
- fonctions de détection des erreurs ;
- codes spéciaux.

Ces indicateurs seront décrits par la suite, d'un point de vue qualitatif, car le mode d'emploi correct et les codes à adopter dépendent du Compilateur particulier qu'on utilise.

EXEMPLE DE COMPILATION (1)

Cette page présente le listing d'un programme simple contenant trois erreurs.
On va tout d'abord lancer l'exécution du programme sans le compiler, en utilisant l'interpréteur Basic (> MBASIC).

```
LIST
10 ' **ESSAI DE COMPILATION**
20 DIM A(5),AS(6)
30 PRINT "ENTREZ LES VALEURS DU TABLEAU (A)"
40 FOR I=1 TO 5
50 PRINT "I=";
60 INPUT "VALEUR ";A(I)
70 NEXT I
80 FOR I=1 TO 5:PRINT A(I):NEXT I
90 '
100 PRINT "ENTREZ LES VALEURS DU TABLEAU (AS)"
110 FOR I=1 TO 6
120 PRINT "I=";
130 INPUT "VALEUR ";AS(I)
140 NEXT I
150 FOR I=1 TO 6:PRINT AS(I):NEXT I
160 '
170 IF A(3)=1743 GOTO 2000
180 GOTO 30
190 END
Ok
RUN
ENTREZ LES VALEURS DU TABLEAU (A)
I= VALEUR ? %
```

En lignes 20, 130 et 150, on référence un tableau nommé AS. Or, ce mot est réservé au système, comme nous l'avons vu lors de l'attribution d'une mémoire tampon E/S pour la gestion d'un fichier disque (par exemple FIELD 1,20 AS BXS...) ; il ne peut donc pas s'appliquer à une variable. Dans certaines versions de Basic interprété, le système est capable de déterminer si un mot réservé est employé indûment.

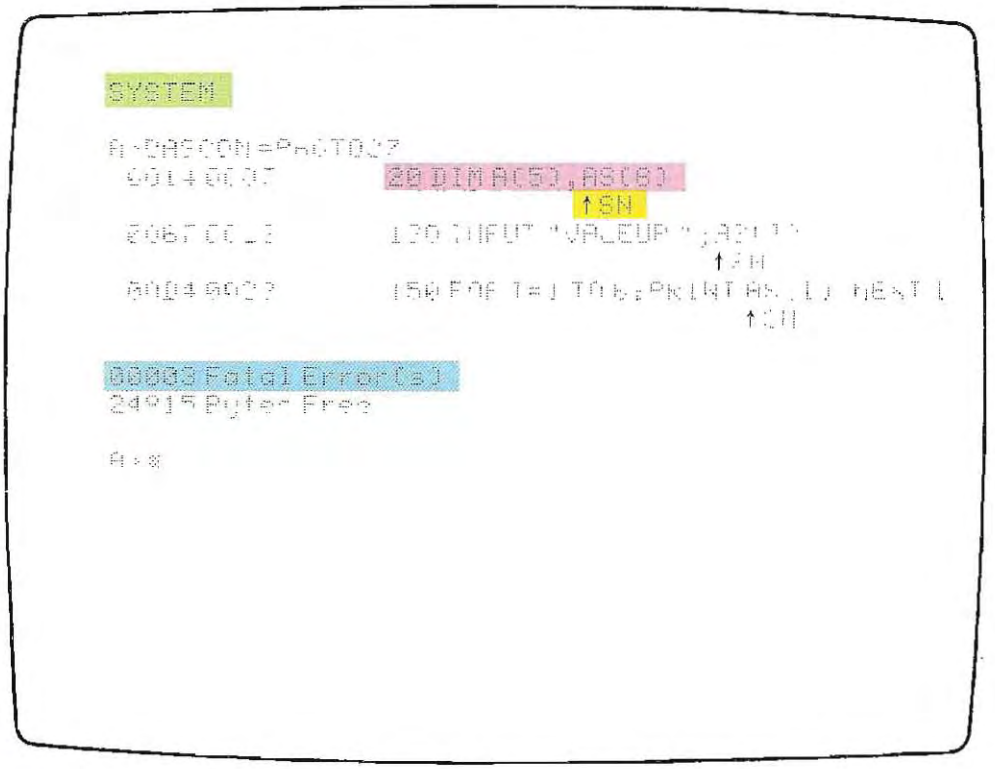
La deuxième erreur est due à l'absence de l'instruction OPTION BASE 1. La numérotation des éléments des tableaux A et AS commence donc à 0 ; les tableaux étant dimensionnés respectivement à 5 et 6, il sera impossible d'affecter une valeur aux éléments A(5) et AS(6).
Dans certains cas, le Basic corrige automatiquement ce type d'erreurs, sans les signaler.

La dernière erreur réside dans l'instruction 170, qui se réfère à une ligne 2000 inexistante. L'interpréteur ne pourra détecter l'erreur qu'au moment de l'exécution de la ligne 170. De par la structure du programme, il n'y aura erreur que si l'opérateur entre A(3)=1743.

Les trois dernières lignes de l'écran montrent que le programme, bien que contenant les erreurs citées, a été lancé et que son exécution en mode interprété se déroule d'une manière apparemment normale pour l'instant.

EXEMPLE DE COMPILATION (2)

Cette page retrace l'étape de compilation du programme pris en exemple à la page précédente.



La première ligne représente la commande tapée par l'opérateur, qui demande le passage sous système d'exploitation ; la deuxième appelle le Compilateur.
La compilation, vérifiant la syntaxe des instructions, débute à partir de la première ligne (10) et se poursuit en séquence.

Le compilateur a détecté en ligne 20 un emploi incorrect du mot réservé AS ; il affiche la ligne à l'écran.

A la ligne suivante, le Compilateur désigne d'une flèche le point où l'erreur a été relevée et affiche un symbole indiquant le type d'erreur détectée : SN signifie Syntax Error (erreur de syntaxe).
Le même type d'erreur est détecté aux lignes 130 et 150. L'erreur de la ligne 170 n'a pas été diagnostiquée car elle est masquée par les erreurs précédentes. elle apparaîtra à la première compilation qui suivra la correction des erreurs détectées.

A la fin de la compilation, le nombre d'erreurs relevées s'affiche à l'écran.

Spécification des conventions. Selon la version du Basic dont on dispose, la syntaxe de certaines instructions sera différente.
On peut rencontrer des différences substancielles, en fonction de la convention sur les mots réservés.
Les noms des instructions ou des fonctions Basic sont réservés au système, et ne peuvent servir de noms de variables. Dans certaines formes de Basic, tous les mots doivent être séparés par des blancs, alors que d'autres versions ne l'imposent pas (le compilateur se charge alors de les insérer). Si on n'indique pas au Compilateur la notation en usage, des erreurs surviendront inévitablement. Par exemple, l'instruction FOR I=A TO C, en enlevant les blancs, devient FORI=ATOC ; si le Compilateur utilise une option erronée, l'instruction prendra une signification tout à fait différente que celle de début de boucle, et affectera à la variable FORI la valeur contenue dans la variable ATOC.
En parlant de ce type d'erreurs, signalons une anecdote amusante. En langage Fortran (comparable au Basic) les boucles sont définies par une instruction du genre

DO 100 I=3,10

Le code DO remplace le code FOR et I=3,10 remplace I=3 TO 10. Le nombre 100 se réfère au numéro de la ligne à laquelle la boucle se termine.
En comparant les deux formes de boucle (Fortran et Basic), on obtient :

Fortran	Basic
10 DO 100 I=3,10	10 FOR I=3 TO 10
100 CONTINUE	100 NEXT I

Le compilateur Fortran enlève tous les blancs contenus dans l'instruction ; la ligne 10 devient donc DO100I=3,10 et l'exactitude de son interprétation est liée uniquement à la présence de la virgule entre les indices (3 et 10). Dans un programme chargé de piloter le lancement d'une sonde spatiale, la virgule fut remplacée par un point dans le fichier source et l'instruction se transforma en DO100I = 3.10.
A l'exécution, au lieu d'effectuer la boucle, on affecta la valeur 3.10 à la variable DO100I : ce fut désastreux pour le lancement.*
En Basic, une erreur de ce genre serait impossible à cause de la syntaxe utilisée (présence de l'instruction NEXT qui ferme la boucle). Si le début (FOR...) est interprété de manière erronée, la compilation sera fausse à la ligne NEXT, car le FOR correspondant manquera. Par contre, en Fortran, l'instruction de fermeture de boucle n'existe pas (le code CONTINUE est fictif) ; si l'interprétation de la ligne initiale est inexacte, la boucle disparaît et aucun diagnostic n'est émis.
D'autres options du Compilateur Basic concernent les boucles, les arrondis et la numérotation des lignes.
Lors du déroulement d'une boucle paramétrée, il est possible de trouver une situation particulière, dans laquelle les deux limites coïncident ; par exemple, la boucle suivante débute et prend fin avec l'indice 3 :

```
10 A=3:B=3
20 FOR I=A TO B
```

L'incertitude quant au déroulement probable ou non de la boucle peut être levée au moyen d'une option qui force l'exécution de chaque boucle au moins une fois.
En règle générale, la même option permet aussi de demander une troncature au lieu d'un arrondi pour toutes les conversions en entier de nombres en double précision.
Une autre option très utile permet d'ignorer les numéros de ligne. Avec cette procédure particulière, les lignes peuvent être numérotées dans un ordre quelconque, ou ne pas être numérotées du tout, exception faite de celles auxquelles on se réfère explicitement dans les instructions GOTO et GOSUB.
De cette manière, un programme Basic devient très semblable à son homologue Fortran, il est plus compact et plus lisible.

Fonctions de traitement des erreurs. L'utilisation de l'instruction ON ERROR GOTO implique un système d'édition de liens du programme plus complexe et donc une forme différente de compilation.
L'option de traitement des erreurs doit être spécifiée par des indicateurs particuliers ; le

* D'après « Software Reliability », Wyley & Sons, New York 1976.

Compilateur produit alors un tableau d'adressage, contenant les numéros de ligne du source Basic. De cette manière, si une erreur apparaît pendant l'exécution du programme, le système peut retrouver le numéro de la ligne incriminée.
Dans les programmes compilés normalement, le numéro de ligne disparaît et chaque instruction est identifiée par son emplacement mémoire (adresse). Lors de la détection d'une erreur, si le système n'a pas conservé en mémoire la table de conversion adresses – numéro de ligne, l'adresse physique où s'est produit cette erreur apparaît à l'écran, en notation hexadécimale. Elle ne peut servir à rechercher la ligne source qui a engendré l'erreur, sinon au moyen de calculs fastidieux.
En phase de test des programmes (debugging), on ne saurait trop recommander l'emploi de cette spécification, même si le programme compilé nécessite un espace mémoire plus grand ; une fois la phase de test achevée, on peut de nouveau recompiler le programme sans les indicateurs d'erreurs.
La table associant à chaque numéro de ligne son adresse hexadécimale possède un point d'entrée (entry point, moyen par lequel on accède à une partie du programme ou à un tableau) pour chaque élément. Généralement, les tableaux possèdent un seul point d'entrée, l'adresse en mémoire du premier élément et les autres sont repérés en fonction de leur position par rapport au point d'entrée. Cela n'est pas possible pour les tables de conversion des numéros de ligne, et un point d'entrée pour chaque instruction du programme s'avère nécessaire. Une gestion aussi étendue des adresses mémoire alourdit considérablement le programme et augmente la taille du fichier objet (binaire) ; d'où la nécessité de l'utiliser uniquement dans la phase de mise au point.

Codes spéciaux. Les fonctions principales des indicateurs concernent le choix du type d'Assembleur, l'activation des instructions TRON et TROFF et le choix du type de compilation particulière, lors du transfert du programme sur ROM.
Les instructions TRON et TROFF, exécutées sans problème dans le cas d'un Basic interprété, sont par contre ignorées dans un programme compilé, à moins qu'on ne les ait déclarées expressément.
Un indicateur spécial permet de spécifier sous quelle forme d'Assembleur on désire le listing. L'Assembleur, exception faite de ses caractéristiques principales, est un langage fondamentalement lié au microprocesseur, et en dépend dans sa syntaxe ; ainsi, une même instruction sera traduite sous des formes différentes selon le microprocesseur.

L'édition de liens

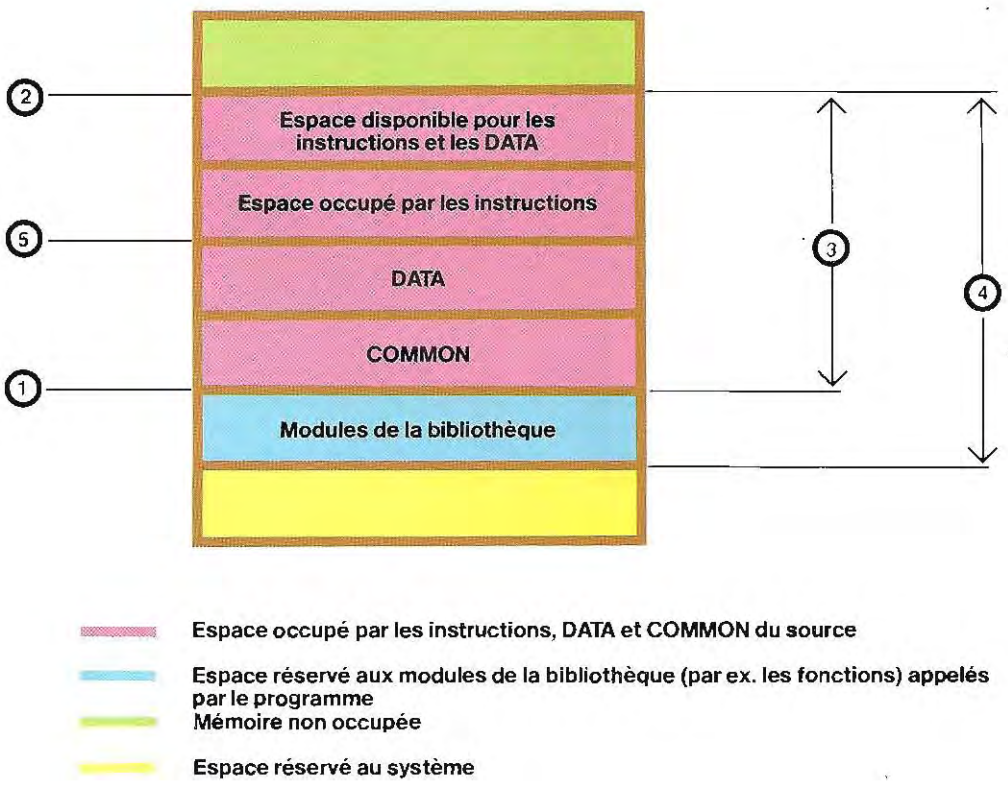
La dernière phase nécessaire à l'obtention d'un programme compilé exécutable est l'édition de liens, ou chaînage des différentes parties. Cette fonction est réalisée par un programme appelé Linking Loader, qui demande en entrée le nom du fichier relogeable (engendré en sortie du compilateur) et génère en sortie le programme exécutable.
Le Linking Loader présente à l'écran une série d'informations concernant l'implantation en mémoire, informations dont les principales sont les suivantes (le numéro se réfère au schéma de la page 732) :

- début du programme (program-start) ; adresse initiale en hexadécimal (**1**) ;
- fin du programme (program-end) ; adresse finale en hexadécimal (**2**) ;
- taille ; nombre d'octets occupés par le programme (**3**) ;
- espace restant (mémoire non utilisée) ;
- adresse de début d'exécution (start-address), exprimée en hexadécimal. On remarquera que l'adresse de début d'exécution du programme ne coïncide pas obligatoirement avec l'adresse de début du programme (**5**) ;
- nombre de pages ; nombre hexadécimal indiquant le nombre de « pages » mémoire de 256 octets utilisées par le programme (**4**).

Le schéma de la page 732 illustre la carte (map) de répartition de la mémoire. On constate que le début du programme comprend l'espace COMMON et l'espace réservé aux valeurs déterminées par l'instruction DATA, alors que l'adresse de début d'exécution coïncide avec la première instruction exécutable. L'espace COMMON est une zone de mémoire accessible à différents programmes, utilisée pour le passage de données et de paramètres de l'un à l'autre.

Des programmes différents peuvent s'appeler tour à tour, mais le chargement en mémoire de l'un recouvre l'espace occupé par l'autre ; ainsi toutes les variables sont réinitialisées, et les valeurs calculées par un programme sont perdues pour le suivant.
Afin de réaliser le transfert des données d'un programme à l'autre, il faut déclarer ces variables comme « communes » (COMMON). Elles seront stockées dans un espace mémoire séparé, qui ne sera pas réinitialisé lors des chargements successifs des programmes.
La zone réservée au système contient tous les pointeurs et autres informations nécessaires au dialogue avec le système d'exploitation.
L'espace dénommé « modules de la bibliothèque » regroupe les fonctions particulières du Basic appelées par le programme d'application. En réalité, on y stocke plus de fonctions que nécessaire, afin de gagner du temps, en évitant une longue opération de recherche et d'édition de liens.
La méthode choisie a été de partager les programmes système en deux bibliothèques : d'une part les fonctions classiques, d'autre part celles qu'on utilise rarement.
Durant la phase d'édition de liens, on charge systématiquement la bibliothèque courante (BRUN.COM), et éventuellement la seconde (BASLIB.REL), seulement si nécessaire.
Ainsi, un programme, même très petit, occupe au moins 16 Koctets de mémoire, qui représentent l'espace minimum nécessaire au module contenant les programmes système.

CARTE DE REPARTITION DE LA MEMOIRE



Test 20

1 / Ce sous-programme de gestion du disque est appelé depuis différents points du programme principal, qui lui transmet la donnée à mémoriser (BF$) et le numéro de l'enregistrement (REC).

```
110 OPEN "R",1,"A:ESSAI",128
120 FIELD 1,128 AS A$
130 LSET A$=BF$
140 PUT 1, REC
```

Quelles erreurs contient-il ?

2 / Les affirmations suivantes sont-elles vraies ou fausses ?
a) Un programme compilé occupe un espace mémoire supérieur au programme source en Basic.
b) Le Compilateur détecte des erreurs qui peuvent échapper à l'Interpréteur.
c) Un programme compilé ne peut pas être copié.

3 / Quelles opérations doit-on exécuter afin d'obtenir un programme exécutable ?

4 / Un Compilateur est-il lié au type d'ordinateur, ou bien peut-il s'utiliser sur n'importe quelle machine ?

Les solutions du test se trouvent page 737.

Instructions particulières complétant le Basic 80

Nous présentons dans ce paragraphe certaines instructions et commandes spécifiques, dont la compréhension demande une très bonne connaissance du langage Basic et de l'architecture de la machine.
Nous allons commencer par quatre instructions d'usage général et nous poursuivrons par l'étude des problèmes liés à la gestion de fonctions graphiques et de certains périphériques spéciaux.

Instructions de portée générale

Dans l'exposé des instructions du Basic, nous avons déjà introduit certaines notions concernant la segmentation, l'enchaînement des programmes et l'appel de sous-programmes. Afin de proposer un tableau complet, il est nécessaire d'évoquer trois autres instructions :

CALL
COMMON
USR

L'instruction STOP sera étudiée par la suite.

CALL. Permet d'appeler une routine écrite en langage Assembleur (sous-programme externe). La syntaxe de l'instruction est la suivante :

CALL Adresse (Arguments)

où Adresse est la variable qui contient l'adresse de départ du sous-programme à appeler, et Arguments représente une liste de variables à lui transmettre afin qu'il puisse les utiliser.
Par exemple, les instructions :

```
10 SUBA=&HA000
20 CALL SUBA(I,X)
```

appellent un sous-programme écrit en Assembleur, qui débute à l'adresse mémoire A000 (le symbole &H indique que la valeur numérique qui suit est hexadécimale), en lui transmettant les valeurs contenues dans les variables I et X. Les données à transmettre doivent avoir été affectées précédemment à

autant de variables. En règle générale, il n'est pas possible de transmettre une valeur constante comme telle : il faut d'abord l'affecter à une variable, puis transférer cette dernière. Par exemple, les instructions permettant le transfert de la valeur 157 à un sous-programme qui débute à l'adresse B010 sont les suivantes :

```
10  V=157
20  ADR=&HB010
30  CALL ADR(V)
```

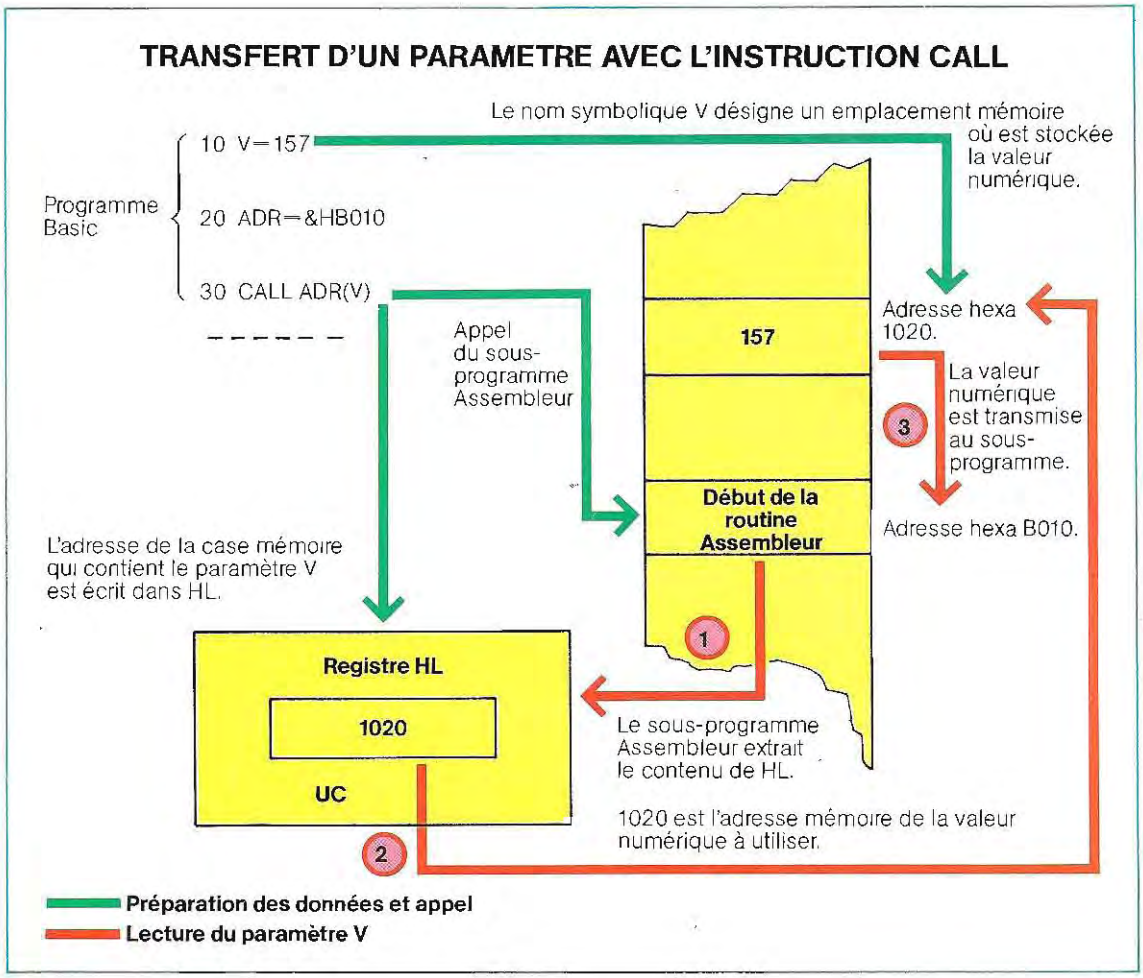
Dans d'autres langages, en particulier le Fortran, le code CALL est similaire au GOSUB du Basic : il appelle un sous-programme interne, écrit dans le même langage que le programme principal. L'instruction CALL ADR(157) donne alors le contrôle à un sous-programme nommé ADR(V), et lui transmet la valeur numérique 157 (argument) qu'il affectera à la variable V.

En Basic compilé, il n'est pas absolument nécessaire de définir numériquement l'adresse de départ du sous-programme appelé : on la désigne par un nom, et l'Editeur de liens, lors du chargement, la recherche dans une table d'adresses.

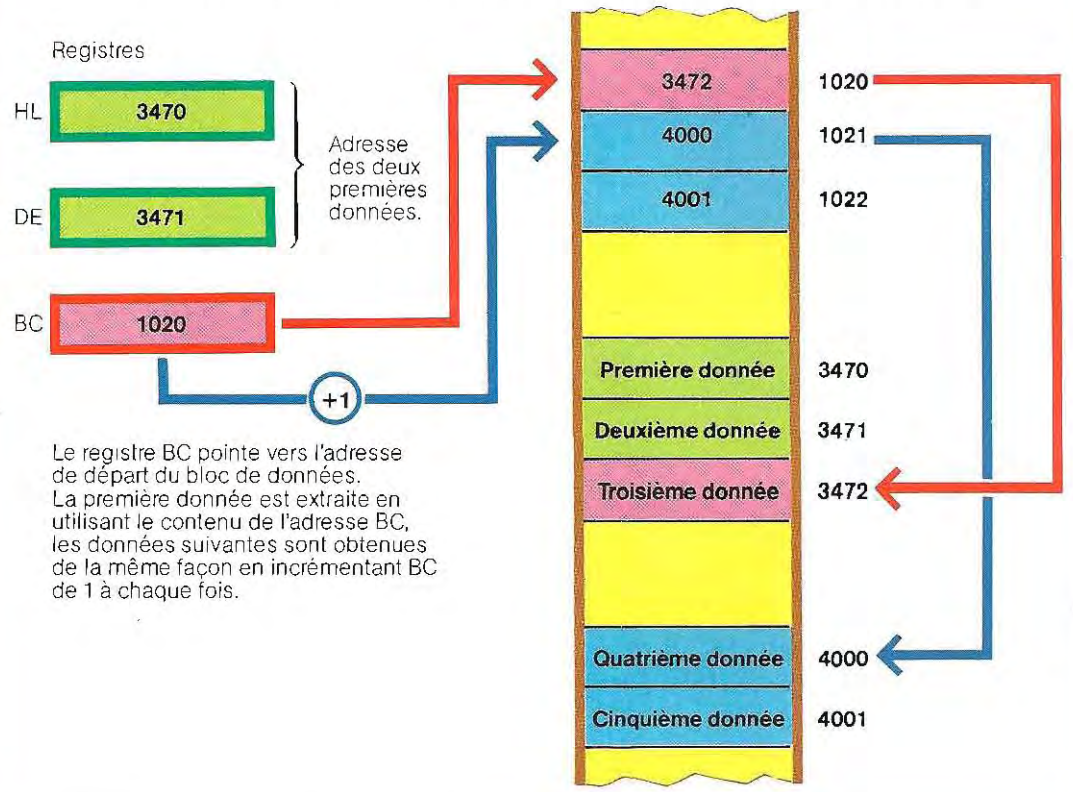
L'instruction CALL permet également d'accéder à des zones de mémoire qui ne peuvent être adressées en Basic : certains Interpréteurs et Compilateurs limitent leur champ d'action à une portion de mémoire bien définie. Dans ce cas, on peut accéder aux deux zones restantes en écrivant un ou plusieurs sous-programmes Assembleur, et en les appelant en Basic par l'instruction CALL.

La transmission des paramètres, qui se formule aisément en langage de haut niveau, présente en Assembleur certaines difficultés, dues au mécanisme complexe de transfert. Les arguments représentent en fait des

TRANSFERT D'UN PARAMETRE AVEC L'INSTRUCTION CALL



SCHEMA LOGIQUE DU TRANSFERT DES PARAMETRES



adresses en mémoire, où se trouvent les valeurs considérées. Chaque adresse est codée sur deux octets ; les paramètres occupent donc toujours deux octets, indépendamment du type de donnée (entier, réel, etc.).
Le transfert des paramètres du Basic à l'Assembleur se déroule selon des modes différents selon leur nombre.
Pour transmettre au plus trois paramètres, on se sert de registres* ; dans le cas contraire, on utilise un bloc de cases mémoire contigües.
Par exemple, lors du transfert de deux paramètres, la première adresse est chargée dans HL** et la seconde dans DE (HL et DE sont les noms de deux registres).
Le programme Assembleur doit prélever le contenu de ces deux registres et les considérer comme des pointeurs (adresse de la case mémoire) vers la valeur de la variable.
L'appel CALL ADR(V) charge donc, dans le registre HL, l'adresse de la case contenant la valeur associée à la variable V ; le sous-programme Assembleur interprète le contenu de HL comme un pointeur et prélève donc la valeur numérique dans la case mémoire indiquée. La page 734 schématise ce procédé.
Le transfert d'un nombre de paramètres supérieur à trois s'avère plus complexe et appelle des sous-programmes système d'allocation de mémoire. En général, les deux premiers paramètres sont transmis à l'aide de deux registres à simple indirection, comme dans le cas précédent, et les autres données sont stockées dans un espace mémoire qui constitue le bloc de données.
Un nouveau registre (BC par exemple) contient un pointeur vers l'adresse de début du bloc (double indirection : BC contient « l'adresse de l'adresse »).
Le schéma ci-dessus détaille la logique mise en œuvre pour accéder aux valeurs.

* Rappelons que les registres sont des mémoires particulières implantées dans l'UC et désignées par des noms bien précis.
** HL est un couple de registres (H + L) utilisés conjointement.

COMMON. Cette instruction permet de transmettre certains paramètres d'un programme remplacé en mémoire par un autre (appelé par CHAIN). La syntaxe est la suivante :

COMMON Liste de Variables

Les variables spécifiées dans la liste sont transférées d'un programme à l'autre et non réinitialisées. Pour transmettre un tableau, il faut ajouter les symboles ( ) à la suite du nom. Par exemple, les instructions

```
10  DIM B(30), E(9)
20  COMMON V,A,B( ),E( ),K$
.....
276  CHAIN "ESSAI",10
```

transmettent au programme ESSAI les valeurs des variables numériques V et A, des tableaux B(30) et E(9), et de la chaîne K$.
En Fortran, un COMMON constitué de plusieurs variables (comme dans l'exemple précédent) peut aussi être désigné par un seul nom. En utilisant une instruction COMMON labellée (comportant un nom générique), le passage des paramètres aux sous-programmes « externes » (en Fortran ou en Assembleur) se simplifie, car il n'est plus nécessaire d'inclure dans l'instruction CALL toute la liste des variables à transférer ; il suffit simplement de citer le com « collectif », défini dans l'instruction COMMON du programme principal. La syntaxe de l'instruction est :

COMMON / NOM / V1, B( ), V3

Les variables V1, V3 et le tableau B sont résumés par le terme NOM. Tous les programmes appelés qui se réfèrent à une instruction COMMON doivent le déclarer expressément. Cette caractéristique du Fortran est offerte également par certaines versions du Basic étendu, pour le chaînage des programmes. Afin d'éviter la répétition des mêmes lignes (une pour chaque programme chaîné), une instruction Basic particulière (% INCLUDE) permet à chacun des programmes appelés d'utiliser les déclarations de l'instruction COMMON, détaillée dans un fichier séparé. De cette manière, la déclaration complète est écrite une seule fois, et chargée à chaque fois qu'elle est nécessaire.

Supposons, par exemple, que les programmes à chaîner soient MAIN, PROG1 et PROG2, mémorisés dans des fichiers distincts (de type .BAS), et utilisent en commun les variables V1, V2, B(30) et A$(6).
Les déclarations de l'instruction COMMON et le dimensionnement des tableaux sont mémorisés dans un fichier séparé, appelé par exemple COMMUN. Pour charger ce fichier dans chaque programme, on utilisera l'instruction

%INCLUDE COMMUN

Les trois programmes auront ainsi la structure suivante :

```
MAIN.BAS (Nom du premier fichier, qui appelle les autres)
10  %INCLUDE COMMUN
.....
165  CHAIN "PROG1"
.....
460  CHAIN "PROG2"

PROG1.BAS              (Premier chaînage)
10  %INCLUDE COMMUN
.....
PROG2.BAS              (Second chaînage)
10  %INCLUDE COMMUN
.....
COMMUN.BAS             (Déclarations)
20  DIM B(30), A$(6)
30  COMMON V1,V2,B( ),A$( ).
```

L'instruction de la ligne 10 provoque l'inclusion, dans chacun des programmes, du contenu du fichier COMMUN ; tout se passera donc comme si les instructions de celui-ci se répétaient dans chacun des programmes.

USR. Définit l'adresse de départ d'un sous-programme en Assembleur. La syntaxe est :

DEF USRN = VALEUR

où le nombre N (de 0 à 9) repère un des sous-programmes définis dans le programme (un numéro est ainsi associé à chaque routine Assembleur), VALEUR étant l'adresse de départ.
Le but de l'instruction USRN est de définir, à l'intérieur du programme Basic, les sous-pro-

Solutions du test 20

1 / Il y a deux erreurs. La première est fondamentale et ne permet pas l'exécution du programme ; la seconde ne se produit que sous certaines conditions.
Première erreur : l'instruction 110 (OPEN) s'exécute à chaque appel du sous-programme. La première fois, elle s'exécute correctement et a pour effet l'ouverture du fichier (ou sa création) ; la fois d'après, par contre, elle provoque une erreur système, avec arrêt de l'exécution du programme, car elle tente d'ouvrir un fichier déjà ouvert.
Seconde erreur : en phase de lecture ou d'écriture sur disque, il est bon de vérifier que le numéro d'enregistrement (REC, instruction 140) est compris entre 1 et le nombre maximal prévu pour le fichier. Avant l'appel au sous-programme, on pourrait en effet rencontrer, par erreur, l'affectation REC=0. En ce cas, l'exécution s'arrêterait car l'enregistrement 0 n'existe généralement pas (sur certaines machines, cependant, les fichiers débutent avec l'enregistrement numéro 0). D'autre part, si on affectait à la variable REC une valeur supérieure au nombre maximum d'enregistrements contenus dans le fichier, on irait écrire sur un espace du disque n'appartenant pas au fichier, entraînant ainsi la destruction d'autres données.

2 / a) faux ; b) vrai. L'affirmation c) doit s'interpréter ainsi : un programme compilé ne peut être copié en tant que source, mais peut être dupliqué sur un autre disque. Par des procédés spéciaux, on peut protéger un programme en interdisant sa copie, ou en la limitant. Il s'agit de méthodes très complexes, réservées uniquement à des programmes réellement secrets.

3 / Voici la succession des opérations à exécuter :
1 - écriture du source Basic et mémorisation sur disque en format ASCII ;
2 - lancement du Compilateur ;
3 - lancement de l'Editeur de liens.
On obtient un programme exécutable mémorisé dans un fichier disque. Pour l'exécuter (sous système d'exploitation), il suffit d'entrer son nom.

4 / Le Compilateur est étroitement lié à la machine, ou mieux, à l'UC utilisée.

grammes Assembleur comme s'ils étaient des fonctions. DEF USRN est comparable à DEF FNX : cette dernière définit des fonctions en Basic, alors qu'USRN appelle un sous-programme Assembleur.
Par exemple, l'instruction DEF USR3=1740, définit le sous-programme Assembleur qui débute à l'adresse mémoire 1740, et lui associe le numéro 3. On l'appelera ensuite comme s'il s'agissait d'une fonction utilisateur. L'instruction X=USR3(K) affecte à X le résultat du calcul exécuté par le sous-programme Assembleur 3, en utilisant le paramètre K.
L'argument (K) peut être de type quelconque (numérique ou chaîne). On indique au sous-programme Assembleur le type dont il s'agit grâce à un registre particulier (registre A), contenant un code lié au type d'argument (par exemple, la valeur 2 désigne un nombre entier, la valeur 3, une chaîne et ainsi de suite). On accède à la valeur de l'argument (valeur de K) de deux façons différentes : pour les valeurs numériques, un couple de registres indique l'adresse mémoire de la valeur ; pour les chaînes, deux autres registres contiennent l'adresse du descripteur de la chaîne.
En général, ce bloc de description est représenté par 3 octets, le premier indiquant la longueur de la chaîne en caractères (de 0 à 255), tandis que les deux autres contiennent l'adresse de début de l'espace mémoire où se trouvent les valeurs (caractères de la chaîne).

S'il s'agit d'une donnée numérique, les registres donnent directement l'adresse mémoire qui la contient et la valeur de l'argument est prélevée en un seul passage. Pour les chaînes, il faut une lecture de plus, puisque les registres contiennent l'adresse du descripteur où est mémorisée l'adresse de la donnée. En règle générale, le bloc de description des chaînes et les chaînes elles-mêmes sont stockés dans l'espace mémoire utilisateur; en revanche, les données numériques sont mémorisées dans des emplacements particuliers appelés Floating Point Accumulator (FAC), et les registres d'adressage pointent sur l'accumulateur particulier impliqué dans l'échange.
Dans certaines versions du Basic, les sous-programmes Assembleur ne peuvent être définis au moyen de l'instruction DEF USRN. Dans ce cas, des cases mémoire réservées reçoivent l'adresse de début avant utilisation de chaque sous-programme. Le nom symbolique de ces cases mémoire dépend de la version particulière du Basic. Pour appeler un sous-programme, on écrit d'abord son adresse de début, grâce à l'instruction POKE, dans la case mémoire réservée à cet effet. Ces versions du Basic ne permettent pas le choix du sous-programme par spécification de son numéro: le sous-programme dont l'adresse de début a été écrite avec l'instruction POKE sera appelé; pour exécuter un autre sous-programme, il faut d'abord opérer le transfert de son adresse de début.
A titre d'exemple, le programme listé page 739 utilise les instructions POKE et PEEK pour écrire deux symboles couleurs choisis parmi 5 (ligne 90) et relire à nouveau, pour chacun, la couleur sélectionnée. Le programme tourne sur COMMODORE VIC-20, et les adresses mentionnées ne sont valables que pour cette machine.

STOP. A pour fonction d'arrêter l'exécution du programme et de retourner en mode commande de la même manière que l'instruction END. La différence réside dans un message émis par le système, qui signale le numéro de ligne où l'instruction STOP a été rencontrée.
L'usage le plus répandu de cette instruction est de quitter l'exécution du programme en cas d'erreurs graves.
A la rencontre de l'instruction STOP, si l'exécution du programme est suspendue, les fichiers utilisés éventuellement ne sont pas refermés (contrairement à ce qui arrive avec l'instruction END); l'exécution du programme peut reprendre à partir de la ligne suivante en entrant la commande CONT (continuer).
L'arrêt momentané du programme permet d'analyser la situation qui se présente au moyen de commandes interactives, puis de reprendre l'exécution après correction éventuelle. Supposons, par exemple, qu'un programme exécute certaines divisions, en utilisant comme diviseurs les variables D1, D2 et D3, calculeés dans le programme lui-même. Il faut alors vérifier que les variables (diviseurs) ne sont pas nulles. Une erreur dans un calcul des diviseurs conduirait à d'autres erreurs dans le déroulement des instructions suivantes. On peut effectuer les contrôles nécessaires par les instructions IF et STOP, par exemple de la manière suivante:

```
IF D1=0 THEN STOP
IF D2=0 THEN STOP
IF D3=0 THEN STOP
```

A la rencontre d'une de ces erreurs, le déroulement du programme s'arrête, à l'écran apparaît le numéro de la ligne contenant le STOP exécuté, et le contrôle est rendu au système. Dans ces conditions, on peut procéder en mode interactif à l'analyse du contenu de toutes les variations qui ont participé au calcul du diviseur erroné, et relancer par la suite l'exécution du programme, afin d'examiner l'effet de cette erreur sur les calculs suivants.
De même, on peut affecter une valeur fictive au diviseur erroné avant de poursuivre l'exécution. Sous ce mode, les corrections du programme lui-même ne sont pas possibles.

Instructions de tracés graphiques

Plusieurs formes de Basic possèdent des instructions particulières pour la présentation de tracés graphiques. Le sujet sera traité en profondeur par la suite: dans ce chapitre, nous n'aborderons que l'étude des principales instructions graphiques, sans détailler leur emploi.
En mode graphique, l'écran est divisé en pixels (picture elements, éléments d'image). La « résolution » la plus répandue est de 512 pixels horizontaux et 256 pixels verticaux: à la

EXEMPLE D'UTILISATION DE PEEK ET POKE

```
0 REM: PROGRAMME ILLUSTRANT L'UTILISATION DE PEEK ET POKE
7 PRINT " ■ ": POKE 36879,29
10 DIM VCOL(5),NCOL$(5)
20 FOR I=1 TO 5
30 READ VCOL(I)
40 NEXT I
50 DATA 0,2,5,7,8
60 FOR I=1 TO 5
70 READ NCOL$(I)
80 NEXT I
90 DATA NOIR, ROUGE, VERT, JAUNE, BLEU
100 INPUT "PREMIERE COULEUR ";COL$
110 K1=0
120 FOR I=1 TO 5
130 IF COL$=NCOL$(I) THEN K1=I
140 NEXT I
150 IF K1=0 THEN PRINT "ERREUR": GOTO 100
160 N1=VCOL(K1)
170 INPUT "SECONDE COULEUR";COL$
180 K2=0
190 FOR I=1 TO 5
200 IF COL$=NCOL$(I) THEN K2=I
210 NEXT I
220 IF K2=0 THEN PRINT "ERREUR": GOTO 170
230 N2=VCOL(K2)
240 POKE 8040,83
250 POKE 38760,N1
260 POKE 8044,83
270 POKE 38764,N2
275 PRINT "CONTROLE COULEUR"
280 INPUT "COULEUR (1/2)";NC
290 IF NC=1 THEN A=PEEK(38760)
300 IF NC=2 THEN B=PEEK(38764)
310 IF A=N1 THEN PRINT "COULEUR= " NCOL(K1): GOTO 330
320 IF B=N2 THEN PRINT "COULEUR= " NCOL(K2)
330 END
```

place des 80 colonnes de caractères, il est possible d'adresser 512 points écran et, à la place des 24 lignes de texte, 256 points écran. L'instruction

HPLOT 10,5

allume sur l'écran le pixel de coordonnées (10,5). La représentation d'un segment de droite s'obtient en déplaçant le curseur du début à la fin du segment, les coordonnées des deux extrémités étant exprimées en pixels. Par exemple, les instructions

LINE (10,5)–(70,100)
HPLOT 10,5 TO 70,100

génèrent toutes deux le tracé d'un segment qui débute en coordonnées (10,5) et se termine au point (70,100).
Par une succession appropriée d'instructions de ce type, on peut dessiner les figures les plus diverses ; un rectangle, par exemple, sera tracé par quatre appels successifs.
Sur certains systèmes, une succession d'appels peut être mémorisée sur disque en format binaire, afin d'être rappelée dans son ensemble au moyen d'une seule instruction. L'instruction DRAW "n" provoque l'affichage sur l'écran du « fichier graphique » numéro n, mémorisé précédemment. Ce fichier peut contenir des figures très complexes, obtenues grâce à de nombreux déplacements élémentaires.
Presque tous les systèmes utilisant cette méthode prévoient des instructions pour le changement d'échelle, le déplacement du dessin ou la coloration de parties de l'écran. En ce qui concerne l'utilisation de ces instructions, présentes (sous des syntaxes différentes) sur presque toutes les machines, certains programmes d'application seront présentés dans le chapitre consacré au dessin sur ordinateur.

Instructions particulières

Grâce à son extrême simplicité, le langage Basic est implanté sur tous les micro-ordinateurs et ordinateurs domestiques. Sa large diffusion a incité les constructeurs à développer des interfaces spéciales, accompagnées de leurs logiciels de gestion, pour offrir l'accès aux périphériques les plus divers.
Ces périphériques, nés à l'origine avec les jeux vidéo, ont été par la suite utilisés par les professionnels, devenant ainsi un moyen de communication excellent entre l'utilisateur et la machine.

Synthétiseur. La génération de sons est un sujet vaste et complexe, qui va de la simple émission d'un « bip » jusqu'à la synthèse vocale, qui permet à l'ordinateur d'émettre des phrases dans des langues différentes. Les applications plus complexes, qui impliquent la présence d'un hardware spécialisé, seront illustrées par la suite. Dans ce chapitre, ne sont considérées que certaines instructions Basic, qui permettent d'engrendrer des sons ou de la musique.
La syntaxe des instructions varie d'un système à l'autre ; on peut toutefois distinguer deux catégories principales : les machines qui possèdent des sous-programmes système spécifiques, pour lesquelles des instructions en langage évolué existent et les machines qui demandent l'utilisation de l'instruction PEEK ou du langage Assembleur.
Cette distinction a pour origine la lenteur d'exécution des instructions en Basic interprété. On génère un son en envoyant les signaux opportuns au haut-parleur. Plus le son est aigu, et plus grande doit être la fréquence* du signal. Si on dépasse une certaine valeur, la fréquence demandée ne peut plus être réalisée, l'Interpréteur n'étant pas assez rapide. Dans ce cas, il faut mettre en œuvre des sous-programmes Assembleur (la génération de sons en Assembleur sera traitée dans le chapitre consacré à ce langage).
Si le système ne possède pas les instructions Basic nécessaires, on peut utiliser l'instruction PEEK. Beaucoup d'ordinateurs domestiques disposent d'une case mémoire spéciale, utilisée comme « interrupteur » du haut-parleur. Chaque fois qu'on accède à cette adresse, il y a émission d'un signal. Pour obtenir un son d'une hauteur déterminée (un « do » par exemple), il faut exécuter une boucle qui active périodiquement et pendant un certain temps, cette case associée au haut-parleur. En modulant correctement les temps, on engendre le son voulu, toujours dans des limites de fréquences assez basses.
On emploie la même méthode pour les fréquences plus élevées, mais avec un programme écrit en Assembleur.
Sur certains systèmes, il existe des programmes utilitaires qui peuvent être appelés par le Basic sous forme d'instruction. Par exemple, l'instruction

SOUND f,d

engendre un son de fréquence f (généralement comprise dans la bande 35 à 32 000 Hz) et de durée d (en secondes).
PLAY est une instruction plus complexe, par laquelle on spécifie différents paramètres (étroitement liés aux connaissances musicales du programmeur), qui permet de jouer de véritables morceaux de musique.

Joystick et paddle. Ces mots évoquent inévitablement les jeux vidéo ; en réalité, il s'agit de fonctions particulières gérant des interfaces grâce à des convertisseurs de déplacement, qui peuvent s'avérer utiles à de nombreuses applications.
Un convertisseur analogique/numérique est un dispositif capable de transformer une grandeur physique donnée en un signal électrique proportionnel à celle-ci.
Les convertisseurs peuvent être d'une autre nature : par exemple, le thermomètre constitue un convertisseur puisqu'il transforme les températures en longueurs (hauteur de la colonne de mercure).
L'ordinateur nécessite des convertisseurs analogique/numérique (A/N) : afin de pouvoir être interprétée par les circuits de la machine, une grandeur physique quelconque (analogique) doit être préalablement convertie en signal électrique (numérique).

* La fréquence d'un signal périodique représente le nombre d'impulsions qui se succèdent en une seconde et se mesure en Hz. Par exemple, à 35 Hz, 35 impulsions ont lieu en une seconde ; chacune a une durée de 1/35 de seconde (cette grandeur s'appelle période).

Le type le plus simple de convertisseur A/N est le **paddle** (levier), constitué par un capteur générant en sortie un signal proportionnel à un déplacement. La fonction Basic correspondante est PDL (n), qui renvoie un nombre entier, compris dans l'intervalle [0,255], et indiquant la « quantité » de déplacement effectué. Le nombre n désigne le port d'entrées/sorties auquel est relié le convertisseur. Par exemple, l'instruction

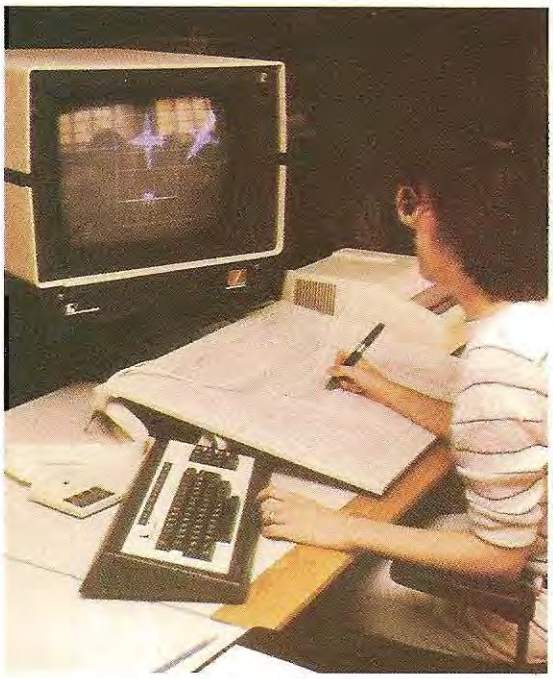
V = PDL(2)

affecte à V la valeur transmise par le paddle relié au port 2. Une fois lue, cette valeur peut être utilisée pour positionner un symbole à l'écran.
En exécutant de manière itérative (boucle) les opérations de lecture et de positionnement du curseur, on reproduit à l'écran le déplacement du levier du paddle de manière presque continue.
Le paddle permet des déplacements le long d'un axe unique, reproduits comme des déplacements du curseur le long d'une seule ligne de l'écran.
Pour un déplacement selon deux directions (sur tout le plan de l'écran), il faut utiliser le **joystick,** sorte de levier bi-directionnel, se comportant comme deux paddles, l'un disposé selon l'axe horizontal, l'autre selon l'axe vertical.
En rappelant que l'aspect formel peut différer d'une machine à l'autre, les fonctions essentielles de contrôle du joystick sont les suivantes :

STICK(n). Fournit les coordonnées de la fonction joystick, sur l'axe n. Par exemple, STICK(0) fournit le déplacement le long de l'axe horizontal, STICK(1) le long de l'axe vertical. Les coordonnées du point d'arrivée s'obtiennent donc avec les instructions

X = STICK(0)
Y = STICK(1)

STRING. Renvoie l'état du bouton (switch) généralement inclus sur le joystick (celui qui permet de « tirer » dans les jeux vidéo). En règle générale, l'état est indiqué par – 1 si le bouton est pressé et 0 en cas contraire. Sur certaines machines est prévu un paramètre [l'instruction devient STRING(n)], pour utiliser plusieurs joysticks, ou pour déterminer s'il existe des demandes en suspens (formulées avant l'exécution de l'instruction), ou encore pour désactiver ou réactiver le joystick (STRING OFF, STRING ON).



K. Reese/Marka

Emploi du stylo optique en CAO. L'opératrice construit la maquette d'un F15.

ON STRING... Cette extension de l'instruction précédente permet d'appeler un sous-programme en fonction de l'état du switch. Par exemple, ON STRING(2), GOSUB 1000 appelle le sous-programme 1000 sur pression du bouton 2.

Le stylo optique. Il s'agit d'un accessoire utilisé en liaison avec les terminaux graphiques, pour l'acquisition de points depuis une tablette graphique. Certains Basic proposent des instructions consacrées à l'activation du stylo et à l'acquisition des coordonnées. Les principales sont :

PEN ON active le stylo.
PEN OFF annule l'instruction précédente.
Z = PEN(N) acquiert les coordonnées. La valeur restituée dans la variable Z dépend de la valeur imposée au paramètre N. Avec N = 1, Z fournit la coordonnée X, avec N = 2 la coordonnée Y. Les valeurs possibles de N et les fonctions correspondantes dépendent étroitement du type de machine.

Précis du Basic 80

INSTRUCTIONS ET COMMANDES

	Instruction ou commande	Effet	Exemple
I	AUTO	Numérote automatiquement les lignes du programme	AUTO 10,5
I	CALL	Appelle un programme en Assembleur	CALL SUBA
I	CHAIN	Enchaîne un programme en lui transmettant éventuellement des variables	CHAIN "TEST"
C	CLEAR	Initialise à zéro toutes les variables	CLEAR
C	CLOAD	Charge un programme résidant sur cassette	CLOAD "PROG"
I	CLOSE	Ferme un fichier	CLOSE 2
I	COMMON	Transmet les variables entre programmes (avec CHAIN)	COMMON A(10)
C	CONT	Relance l'exécution d'un programme (après STOP)	CONT
C	CSAVE	Sauvegarde un programme sur cassette	CSAVE "PROG"
I	DATA	Définit les valeurs des constantes	DATA 3, 7, 9
I	DEF FN	Définit des fonctions utilisateur	DEF FNX=A+B
I	DEFINT	Déclare des variables entières	DEFINT K
I	DEFSNG	Déclare des variables en simple précision	DEFSNG R
I	DEFDBL	Déclare des variables en double précision	DEFDBL Z
I	DEFSTR	Déclare des chaînes de caractères	DEFSTR A
I	DEF USR	Définit l'adresse de début d'un programme en Assembleur	DEF USR2=1275
C	DELETE	Efface des lignes de programme	DELETE 5-25
I	DIM	Dimensionne des tableaux	DIM A(3), B$(10)
C	EDIT	Passe en mode éditeur pour corriger le programme	EDIT 20
I	END	Termine un programme	END
C	ERASE	Supprime les tableaux que l'on peut ainsi redéfinir	ERASE A, B$
I	ERR	Variable système pour la gestion des codes d'erreur	IF ERR=3 THEN...
I	ERL	Variable système pour signaler la ligne qui engendre l'erreur	IF ERL=127 THEN...
I	ERROR	Simule une erreur	ERROR 3
I	FIELD	Attribut une ou des mémoires tampon d'E/S à un fichier sur disque	FIELD 2,5 AS C$
I	FOR... NEXT	Boucle	FOR I=1 TO 100 NEXT I
I	GET	Lecture d'un enregistrement	GET 1,56
I	GOSUB	Appelle un sous-programme interne	GOSUB 5740
I	GOTO	Saute à la ligne spécifiée	GOTO 250
I	IF... THEN	Instruction conditionnée	IF K=1 THEN...
I	INPUT	Saisie de données au clavier	INPUT "données";A,B
I	INPUT #	Saisie de données depuis un fichier séquentiel	INPUT # 1,A

	Instruction ou commande	Effet	Exemple
C	KILL	Efface un fichier sur disque	KILL "données"
I	LET	Affecte une valeur à une variable	LET A=3+B
I	LINE INPUT	Saisie d'une ligne entière de caractères tapée au clavier	LINE INPUT A$
I	LINE INPUT #	Lecture d'une ligne de caractères depuis un fichier séquentiel	LINE INPUT # 1,A$
C	LIST	Affiche sur l'écran le programme résidant en mémoire centrale	LIST 10-50
C	LLIST	Liste le programme sur imprimante	LLIST 21-90
C	LOAD	Charge un programme à partir du disque	LOAD "A:TEST"
I	LPRINT	Imprime les variables spécifiées	LPRINT A,B,V
I	LPRINT USING	Imprime avec formatage	LPRINT USING "#.#";A
I	LSET	Transfère les données avec justification à gauche	LSET A$=C$
C	MERGE	Adjoint un programmme sur disque au programme en mémoire	MERGE "A:PROG"
C	NAME	Renomme un fichier sur disque	NAME "TEST"AS "NOUV"
C	NEW	Efface le programme contenu en mémoire centrale	NEW
C	NULL	Détermine le nombre d'espaces à écrire à la fin de chaque ligne	NULL 5
I	ON ERROR...	Permet la gestion par programme des erreurs système	ON ERROR GOTO 20
I	ON... GOTO	Exécute un saut conditionné par la valeur d'un paramètre	ON K GOTO 10, 20...
I	ON... GOSUB	Appel d'un sous-programme conditionné par un paramètre	ON L GOSUB 1000, 2000...
I	OPEN	Ouvre un fichier sur disque	OPEN "I", 2, "FISEQU" OPEN "R", 1, "FIRAND", 50
I	OPTION BASE	Définit la première valeur des indices (1 ou 0)	OPTION BASE 1
I	OUT	Transfère un octet sur le port de sortie	OUT 2, 170
I	POKE	Ecrit un octet dans la case mémoire spécifiée	POKE 2320, 170
I	PRINT	Affiche sur écran	PRINT A,B,C$
I	PRINT USING	Affiche avec formatage	PRINT USING "#.#";A
I	PRINT #	Ecrit dans un fichier séquentiel	PRINT # 3,V,C$
I	PRINT # USING	Ecrit dans un fichier séquentiel avec formatage	PRINT # 3 USING "#.#;A
I	PUT	Ecrit un enregistrement dans un fichier en accès direct	PUT 3,26
I	RANDOMIZE	Active le générateur de nombres aléatoires	RANDOMIZE
I	READ	Saisit les valeurs définies par un DATA	READ K,L,N$
I	REM	Permet l'insertion de commentaires. Sur certains Basic, il est remplacé par le symbole ', sur d'autres par ! ou *	10 REM * Comment. *
C	RENUM	Renumérote un programme résidant en mémoire	RENUM 10,20,5
I	RESTORE	Positionne le pointeur au début d'une ligne de DATA	RESTORE 1475

	Instruction ou commande	Effet	Exemple
I	RETURN	Rend le contrôle au programme appelant	RETURN
I	RESUME	Relance l'exécution d'un programme après une erreur	RESUME 5
I	RSET	Transfère les données avec justification à droite	RSET A$=B$
C	RUN	Lance l'exécution d'un programme	RUN
C	SAVE	Sauvegarde un programme sur disque	SAVE "A:TEST"
I	STOP	Suspend l'exécution d'un programme et retourne en mode commande	STOP
I	SWAP	Echange le contenu de deux variables	SWAP A$,B$
I	TRON	Active le mode trace (visualisation des instructions exécutées)	TRON
I	TROFF	Désactive TRON	TROFF
I	WAIT	Provoque une attente jusqu'à l'arrivée d'une valeur donnée sur le port spécifié	WAIT 2,240
I	WHILE... WEND	Exécute une boucle tant qu'une condition est vraie	WHILE K WEND
I	WIDTH	Indique à l'imprimante le nombre maximum de caractères par ligne	WIDTH 60
I	WRITE	Ecrit sur un terminal	WRITE A,B
I	WRITE #	Ecrit sur un fichier séquentiel	WRITE # 2,A,B,

FONCTIONS

Fonction	Description	Exemple
ABS(V)	Fournit la valeur absolue	ABS(– 3 * 5)
ASC(A$)	Fournit le code ASCII du premier caractère de la chaîne	ASC(A$)
ATN(A)	Arc-tangente de A	ATN(2)
CDBL(R)	Convertit en double précision	CDBL(37.4)
CHR$(N)	Fournit le caractère ASCII correspondant à la valeur numérique décimale N	CHR$(68)
CINT(K)	Convertit K en nombre entier pas arrondi	CINT(4.75)
COS(A)	Calcule le cosinus de l'angle A (en radians)	COS(2.1)
CSNG(D)	Convertit le nombre D en simple précision	CSNG(7.569114)
CVI(A$)	Convertit la chaîne en nombre entier	CVI(A$)
CVR(A$)	Convertit la chaîne en nombre réel	CVR(A$)
CVD(A$)	Convertit la chaîne en nombre en double précision	CVD(A$)
EOF(N)	Renvoie – 1 quand on rencontre la fin d'un fichier séquentiel	IF EOF(1)...
EXP(N)	Calcule l'exponentielle e^N	EXP(3.5)
FIX	Tronque la valeur X à sa partie entière	FIX(57.921)
FRE(N)	Fournit le nombre d'octets encore disponibles en mémoire centrale	FRE(0)
HEX$(M)	Fournit une chaîne qui est la représentation hexadécimale de M	HEX$(75)

Fonction	Description	Exemple
INKEY$	Saisie d'un caractère entré au clavier	A$=INKEY$
INP(N)	Saisie d'un octet en provenance du périphérique relié au port N	A=INP(4)
INPUT$(N)	Saisie de N caractères au clavier (sans les visualiser)	B$=INPUT$(2)
INSTR(A$,B$)	Recherche la chaîne A$ dans B$ et en fournit la position	K=INSTR(A$,B$)
INT(L)	Renvoie l'entier <=L	I=INT(87.3)
LEFT$(A$,N)	Prélève les N premiers caractères de A$	B$=LEFT$(A$,4)
LEN(A$)	Donne la longueur de la chaîne A$	K=LEN(A$)
LOC(N)	Donne le numéro du dernier enregistrement lu ou écrit dans un fichier à accès direct	N=LOC(1)
LOG(K)	Calcule le logarithme népérien de K	R=LOG(7.21)
LPOS(X)	Renvoie la position de la tête d'impression	N=LPOS(A)
MID$(I)	Extrait une partie de chaîne	E$=MID$(A$,3)
MID$(A$,I,N)	Prélève N caractères de A$ à partir de la position I	B$=MID$(A$,5,3)
MKI$(V)	Convertit l'entier V en une chaîne de deux octets	A$=MKI$(2)
MKS$(V)	Convertit V (simple précision) en une chaîne de quatre octets	A$=MKS$(71.5)
MKD$(V)	Convertit V (double précision) en une chaîne de huit octets	A$=MKD$(1521.76)
OCT$(N)	Renvoie la représentation hexadécimale de N	A$=OCT$(12)
PEEK(A)	Renvoie l'octet contenu à l'emplacement de mémoire A	PEEK(&H4701)
POS(N)	Donne la position du curseur	L=POS(1)
RIGHT$(A$,N)	Prélève N caractères de la chaîne A$ à partir de la droite	B$=RIGHT$(A$,7)
RND(K)	Génère un nombre aléatoire entre 0 et 1	R=RND(3)
SGN(A)	Renvoie -1, 0 ou 1 selon le signe de A	S=SGN(-3)
SIN(A)	Calcule le sinus de l'angle A exprimé en radians	R=SIN(3.14)
SPACE$(N)	Crée une chaîne de N espaces blancs	A$=SPACE$(3)
SPC(I)	Transmet I blancs au terminal	PRINT "A",SPC(7)
SQR(V)	Calcule la racine carrée de V	A=SQR(7)
STR$(N)	Convertit le nombre N en une chaîne	A$=STR$(3)
STRING$(N,A$)	Crée une chaîne de N caractères égaux au premier caractère de A$	B$=STRING$(5,A$)
TAB(N)	Avance le curseur à la position N	PRINT "X";TAB(6)
TAN(A)	Calcule la tangente de l'angle A exprimé en radians	Y=TAN(7)
USR(K)	Appelle un sous-programme en Assembleur en lui transmettant l'argument K	A=USR1(X)
VAL(A$)	Essaie d'interpréter A$ comme un nombre et renvoie cette valeur	N=VAL(A$)
VARPTR(A)	Donne l'adresse où est mémorisée la variable A	VARPTR(X)

Les variantes du Basic

Il existe de nombreuses versions du Basic, plus ou moins proches les unes des autres. Cependant, la plupart des instructions et fonctions du Basic 80 sont généralement disponibles malgré certaines variations de leur forme lexicale ou de leur syntaxe. Les différences les plus marquées portent sur les instructions affectant les domaines suivants :

- Gestion des chaînes de caractères
- Gestion des fichiers sur disque
- Gestion des opérations d'entrée/sortie
- Gestion des périphériques
- Transmission et réception des données.

Gestion des chaînes de caractères

Une chaîne peut contenir un nombre de caractères allant de 0 (chaîne vide) à un maximum variable selon les versions du Basic. L'espace mémoire qui lui est réservé dépend alors de ce nombre (allocation dynamique). A tout moment, la longueur d'une chaîne peut être déterminée grâce à la fonction LEN. Toutefois, certains Basic permettent de fixer, pour chaque chaîne, une longueur maximale (en caractères) inférieure à la limite initiale. Ce paramètre est déterminé soit par l'intermédiaire de l'instruction DIM, soit grâce à la fonction ALLOCATE, qui n'est prévue que sur les machines les plus puissantes. Celle-ci autorise d'ailleurs une allocation d'espace mémoire à caractère provisoire. Imaginons, par exemple, qu'un sous-programme nécessite la mémorisation de 1000 chaînes. L'emploi de l'instruction DIM A$ (1000) réserverait bien une zone de mémoire suffisante, mais de façon définitive. Ceci revient à « perdre » cet espace mémoire si les chaînes en question n'ont aucun intérêt pour la suite du programme. Il est préférable, dans ce cas, d'employer l'instruction

```
ALLOCATE A$(1000)
```

car il est possible, une fois les données utilisées, d'invalider cette instruction par la commande DEALLOCATE. Dans le cas présent, la syntaxe serait :

```
DEALLOCATE A$(1000)
```

Avec des Basic aussi complets, il est possible de limiter la longueur des chaînes à un certain nombre de caractères. Ce maximum devra être indiqué entre crochets après les instructions DIM, COM ou ALLOCATE. Voici quelques exemples :

DIM A$[5]	Attribution à la chaîne A$ d'une longueur maximale de 5 caractères
COM A$[5]	Même résultat mais en « COMMON », c'est-à-dire dans une partie de mémoire commune à tous les sous-programmes
ALLOCATE A$[5]	Résultat identique, mais l'espace réservé peut être libéré et, éventuellement, affecté à d'autres variables
DIM B$(7)[20]	Réservation d'espace mémoire pour 7 chaînes de 20 caractères chacune.

Les systèmes sur lesquels cette symbolisation est en vigueur présentent généralement deux caractéristiques très importantes :

- une longueur limite pouvant atteindre 32767 caractères pour les chaînes ;
- la mise en œuvre de sous-chaînes.

Alors qu'en Basic 80, l'extraction de parties de chaînes nécessite l'emploi des fonctions LEFT$, RIGHT$ et MID$, le recours aux crochets facilite considérablement cette tâche. Soit, par exemple, A$ = "Ceci est du Basic". Pour extraire le mot « Basic » de cette chaîne, il faudrait utiliser :

```
B$=RIGHT$(A$,5)          en Basic 80
ou
B$=A$[13]                dans les variantes
                         employant les crochets
```

Cette dernière instruction a pour résultat l'extraction de tous les caractères de A$ situés à partir de la position 13 et leur affectation à B$. La chaîne ainsi créée est une sous-chaîne. On peut préciser jusqu'à quel caractère doit s'effectuer l'extraction.
Ainsi, l'instruction :

```
B$=A$[3,8]
```

commande le transfert dans B$ du contenu

Basic 80	Variantes	Opération exécutée
B$=LEFT$(A$,3)	B$=A$[1,3]	Extraction des trois premiers caractères à partir de la gauche
B$=RIGHT$(A$,2)	—	Extraction des deux derniers caractères à partir de la droite
B$=MID$(A$,4,1)	B$=A$[4;1]	Extraction du quatrième caractère
DIM B$(3)	DIM B$(3)	Réservation d'espace mémoire pour trois chaînes
—	DIM B$(3)[20]	Réservation d'espace mémoire pour trois chaînes d'une longueur de vingt caractères chacune au maximum
—	ALLOCATE B$(3)	Réservation temporaire d'espace mémoire pour trois chaînes de caractères
Non prévue sous cette forme mais l'instruction ERASE a un effet comparable	DEALLOCATE B$(3)	Annulation de l'ALLOCATE correspondant. L'espace mémoire est à nouveau disponible

de A$ de la position 3 à la position 8. On obtient le même résultat par :

B$=A$[3;6]

Cette instruction demande, en effet, l'extraction de 6 caractères à partir de la position 3 de A$, ce qui équivaut bien à prélever les caractères situés de la troisième à la huitième position. On trouvera dans le tableau du haut de cette page une récapitulation des principales instructions de gestion des chaînes de caractères qui sont différentes en Basic 80 et dans les versions du Basic acceptant les sous-chaînes.
Les différences que nous venons de voir sont assez importantes, les autres restant purement formelles. Ainsi, dans certaines versions du Basic, la fonction ASC(A$) sera remplacée par VAL(A$), ou encore STRING$(N) deviendra VAL$(N). Il s'agit évidemment des mêmes fonctions bien qu'elles soient désignées sous une autre forme syntaxique.

Gestion des fichiers sur disque

C'est vis-à-vis des unités de disque que les instructions d'entrée/sortie varient le plus souvent d'un Basic à l'autre. Si la logique de ces accès reste relativement homogène, la syntaxe des commandes peut revêtir des formes extrêmement différentes.
A titre d'exemple, nous allons passer en revue les principales instructions du DOS 3.3 : système d'exploitation de disquettes employé aur Apple et tous les micro-ordinateurs compatibles.

CATALOG — Cette instruction provoque l'affichage de l'ensemble des fichiers présents sur la disquette. C'est l'équivalent de la commande DIR du CP/M.

LOAD, SAVE — Instructions de chargement et de sauvegarde (identiques sous CP/M).

INIT — Instruction de formattage d'une disquette. Homologue de FORMAT, elle présente toutefois une caractéristique particulière. En effet, cette commande doit être exécutée avec un programme présent en mémoire centrale. Celui-ci sera automatiquement sauvegardé sur la disquette après le formattage. Par la suite, ce programme appelé « programme de salutations » sera

chargé et exécuté à chaque mise sous tension de l'ordinateur. L'opérateur n'aura alors aucune commande à entrer : le logiciel se mettant en route directement.
La syntaxe exacte de cette instruction est :

INIT p,Vv,Ss,Dd

où p est le nom sous lequel sera sauvegardé le programme de salutations ;
v est le numéro affecté à la disquette (254 par défaut) ;
s est le numéro du port où est inséré la carte contrôleur des disquettes (6 par défaut) ;
d est le numéro du lecteur de disquettes relié à cette carte (1 par défaut).

DELETE — Instruction de suppression d'un fichier (ERA en CP/M).

LOCK — Instruction de verrouillage qui protège les fichiers de l'écriture ou de la destruction (annulée par UNLOCK).

RENAME — Instruction servant comme en CP/M à rebaptiser un programme ou un fichier.

VERIFY — Instruction servant à vérifier l'intégralité d'un fichier. Certains incidents peuvent occasionner l'altération de tout ou d'une partie d'un fichier. Ce contrôle d'une mémorisation correcte des données utilise une méthode comparable à celle du bit de parité. Pour chaque secteur de la disquette, est mémorisé un « total de contrôle » (en anglais, checksum) calculé en fonction de son contenu. Lorsqu'une donnée est modifiée accidentellement, ce total n'est pas mis à jour. Il n'y a donc plus concordance et cela sera détecté par le système lors de la vérification.

Le DOS et le CP/M distinguent les deux mêmes types de fichiers (accès direct ou séquentiel) ; cependant, ils utilisent une technique très différente pour les opérations de lecture et d'écriture.
Avec le DOS, celles-ci sont respectivement

EXEMPLE DE GESTION DES FICHIERS A ACCES SEQUENTIEL SOUS DOS

```
90 REM : Programme PROG1
100 REM : Opérations d'E/S sur disquette
120 REM : caractere de commande du DOS
140 D$=CHR$(4) : REM CTRL-D
160 REM
180 REM : *** OUVERTURE ***
200 PRINT D$;"OPEN ESSAI"
220 REM
260 INPUT " CHAINE : ";A$ : REM SAISIE AU CLAVIER
280 REM : *** ECRITURE ***
300 PRINT D$;"WRITE ESSAI" : REM PROCHAINES SORTIES VERS LA DISQUETTE
320 PRINT A$
325 PRINT D$ : REM ANNULE LA COMMANDE DOS PRECEDENTE
340 PRINT "DONNEE LUE : "; : REM AFFICHAGE A L'ECRAN
350 REM
360 REM : *** LECTURE ***
380 PRINT D$;"READ ESSAI, B0" : REM PROCHAINES ENTREES DE LA DISQUETTE
400 INPUT B$
410 REM : *** FERMETURE ***
420 PRINT D$;"CLOSE"
460 PRINT B$
480 END
```

K. Reese/Marka

La table traçante, un outil précis et rapide. Ici, les courbes de production de puits de pétrole.

commandées par READ et WRITE. Ces instructions sont envoyées sous forme de chaîne et précédées du code CTRL-D (ou CHR$(4)) qui indique que la chaîne suivante doit être interprétée comme une commande du DOS. Ainsi, une opération de lecture prendra la forme :

```
PRINT CHR$(4);"READ REPERTOIRE"
INPUT NOM$, PRENOM$, TEL$
```

La deuxième instruction indique au système que les prochaines données doivent être saisies sur la disquette. La seconde affecte les variables en conséquence.
Le programme dont le listage se trouve page 748 est destiné à illustrer les méthodes de lecture et d'écriture sur un fichier séquentiel. Il a été écrit sur un micro-ordinateur APPLE IIe, mais il fonctionnerait sur les autres modèles ainsi que sur toutes les machines compatibles Apple. Le caractère CHR$(4) (CTRL-D) doit être transmis avant chaque chaîne contenant une instruction du DOS. On a donc intérêt à l'affecter à une variable (D$: ligne 140). L'ouverture du fichier appelé ESSAI est demandée en ligne 200 par une instruction de sortie (PRINT) qui sera aiguillée vers le DOS à cause du caractère de contrôle. On indique ensuite, de la même façon, qu'on va procéder à une écriture sur la disquette (WRITE : ligne 300). Toutes les données envoyées en sortie seront alors orientées vers la disquette et non plus vers le périphérique précédemment en vigueur (écran, imprimante,...) et ce jusqu'à la prochaine commande DOS (précédée par D$). Les entrées/sorties sur disquette sont donc gérées par les commandes PRINT et INPUT classiques ; seules la destination et la provenance des informations sont modifiées au préalable.
Ainsi, une opération de lecture sera réalisée comme suit :

— Ouverture du fichier (déjà ouvert ligne 200)
— Transmission de la commande de lecture

LECTURE ET ECRITURE DE DONNEES NUMERIQUES EN ACCES SEQUENTIEL

```
10 REM : Programme PROG2
20 REM : FICHIER ACCES SEQUENTIEL
30 REM
100 REM : DONNEES NUMERIQUES
110 INPUT "Nombre de données ";N
120 DIM V(N)
140 PRINT "Entrez les valeurs"
160 FOR I=1 TO N
180 PRINT "Donnée numéro : ";I
200 INPUT V(I)
210 NEXT I
220 REM :
240 D$=CHR$(4)
260 PRINT D$;"OPEN ESSAI2"
280 PRINT D$;"WRITE ESSAI2"
300 FOR I=1 TO N
320 PRINT V(I)
340 NEXT I
360 REM :
400 PRINT D$;"OPEN ESSAI2" : REM Ferme puis réouvre le fichier
410 REM                      et réinitialise le pointeur en début de fichier
420 PRINT D$;"READ ESSAI2"
440 FOR I=1 TO N
460 INPUT V(I)
480 NEXT I
500 PRINT D$;"CLOSE ESSAI2"
520 REM === AFFICHAGE ===
540 PRINT " ** DONNEES LUES **"
560 FOR I=1 TO N
580 PRINT "Donnée N°: ";I,V(I)
600 NEXT I
620 END
```

(ligne 380 où B0 positionne le pointeur en début de fichier : inutile juste après l'OPEN)
- Lecture des données (ligne 400)
- Fermeture du fichier (ligne 420)

Une commande de lecture/écriture peut être invalidée par toute autre commande du DOS, voire même une chaîne vide (D$ seul : ligne 325). Le système est ainsi informé qu'on veut retourner aux unités d'E/S antérieures, généralement la console vidéo. Ainsi, bien que les lignes 320 et 340 soient d'une forme identique, leurs effets seront différents. En effet, la première sortie sera dirigée vers la disquette puisqu'elle intervient dans un processus d'écriture du DOS ; par contre, ce processus ayant été annulé en 325, la seconde sortie provoquera un affichage à l'écran.
En haut de cette page se trouve le listage d'un programme qui effectue d'abord l'écriture, puis la lecture de données numériques dans un fichier à accès séquentiel.
Les opérations d'écriture occupent les lignes 260 à 340. Après ouverture du fichier (ligne 260), on transmet la commande d'écriture, puis les données sont envoyées une à une par l'intermédiaire d'une boucle (lignes 300 à 340).
Le nombre N de ces données et leurs valeurs ont été préalablement saisis au clavier (lignes 110 à 210).
Le transfert donnée par donnée est caractéristique des systèmes qui travaillent sous DOS. Il est du au fait que celui-ci ne reconnaît comme séparateur que le caractère CR (retour chariot = CHR$(13)). Il est donc impératif d'insérer un CR entre chacune des valeurs numériques consécutives.
Dans notre exemple, le recours à une boucle autour de l'instruction PRINT permet d'envoyer chaque donnée séparément et suivie de CR, puisque l'on utilise ni « , » ni « ; ».
Il est aussi possible de transmettre systématiquement CHR$(13). On pourrait effectivement sauvegarder les variables X, Y et Z par l'un ou l'autre des programmes suivants :

1) D$=CHR$(4) Code de commande
CR$=CHR$(13) Retour chariot
PRINT D$;"OPEN ESSAI"
PRINT D$;"WRITE ESSAI"
PRINT D$

2) D$=CHR$(4)
PRINT D$;"OPEN ESSAI"
PRINT D$;"WRITE ESSAI"
PRINT X
PRINT Y
PRINT Z
PRINT D$

La seconde solution fait appel, comme avec l'emploi d'une boucle, à l'envoi automatique de CR à la fin d'une instruction PRINT.

Le DOS possède en outre deux commandes inexistantes sous CP/M. Il s'agit de :

- APPEND
- POSITION

L'instruction APPEND permet de compléter un fichier par adjonction d'enregistrements supplémentaires.

EXTENSION ET LECTURE D'UN FICHIER A ACCES SEQUENTIEL

```
10 REM : Programme PROG4
20 REM ' FICHIER ACCES SEQUENTIEL
30 REM : INSTRUCTION APPEND
35 HOME : REM VIDE ECRAN
100 D$=CHR$(4)
105 ONERR GOTO 1000
110 REM *** OUVERTURE ***
120 PRINT D$;"APPEND ESSAI"
160 PRINT "ENTREZ LES DONNEES"
180 PRINT "Tapez FIN pour terminer"
190 K=0
200 INPUT B$
220 IF B$="FIN" GOTO 320
240 PRINT D$;"WRITE ESSAI"
260 PRINT B$
270 PRINT D$
280 PRINT "La donnée est mémorisée"
290 K=K+1
300 GOTO 200
320 REM : SAISIE TERMINEE
340 PRINT D$;"CLOSE ESSAI"
365 IF K=0 GOTO 960
380 PRINT "IL A ETE AJOUTE : ";K;" ENREGISTREMENTS"
410 PRINT
420 PRINT "LECTURE DES DONNEES"
430 PRINT D$;"OPEN ESSAI"
680 INPUT "Combien y a-t-il d'enregistrements à lire ?";NE
700 INPUT "A partir de quelle position ?";P%
770 PRINT D$;"POSITION ESSAI,R";P%
780 PRINT D$;"READ ESSAI"
800 FOR I=0 TO NE-1
820 INPUT A$
845 IN%=I+P%
860 PRINT "Donnée n° ";IN%,A$
900 NEXT I
920 PRINT D$;"CLOSE ESSAI"
960 END
1000 REM : ** ERREUR **
1020 HTAB 10: VTAB 23: INVERSE : REM INVERSION VIDEO
1030 PRINT " ** FIN DE FICHIER ** "
1055 NORMAL : REM VIDEO NORMALE
1080 GOTO 960
```

Ceci est impossible sous CP/M, qui oblige à récrire intégralement le fichier modifié.
POSITION est une instruction permettant de placer le pointeur d'un certain nombre d'enregistrements. Ce déplacement ne peut s'effectuer que vers l'avant (déplacement relatif) et positionne le pointeur sur le premier caractère de l'enregistrement désigné. Les enregistrements sont reconnus et dénombrés grâce au séparateur (caractère CR).
Le listage reproduit sur cette page montre comment on procède à l'extension d'un fichier à accès séquentiel (APPEND) et à sa lecture, totale ou partielle, à partir d'un enregistrement donné (POSITION).
A la lecture d'un fichier dont on ignore la longueur, il peut arriver qu'on tente d'accéder à un enregistrement inexistant. Le système émet alors un message d'erreur (END OF DATA) et interrompt l'exécution. Cette fin horrible peut être évitée par l'emploi de l'instruction ONERR GOTO nn (identique sous CP/M). En cas d'erreur, le système effectuera un branchement à la ligne de numéro nn où on aura écrit une routine de traitement d'erreurs. Une telle routine peut être générale ou examiner le code d'erreur qui a été mémorisé et prévoir un traitement différent selon chaque cas.
Le programme du bas de cette page montre comment on peut utiliser l'instruction ONERR GOTO ... Suivant le type d'erreur apparu, un code est affecté à l'emplacement de mémoire 222.
L'instruction PEEK (222) fournit donc une valeur numérique correspondant à une erreur particulière que l'on peut traiter en connais-

EXEMPLE D'UTILISATION DE L'INSTRUCTION ONERR (DOS)

```
10 REM : PROGRAMME PROG5
20 REM :
30 REM : EMPLOI de ONERR GOTO
40 REM :
50 ONERR GOTO 1000
60 REM :
80 D$=CHR$(4)
100 PRINT D$;"OPEN ESSAI"
120 PRINT D$;"READ ESSAI"
140 FOR I=1 TO 100
160 INPUT A$
200 PRINT "DONNEE n° :";I,A$
240 NEXT I
260 PRINT D$;"CLOSE ESSAI"
280 PRINT "A LA FIN DU FICHIER"
300 END
1000 REM : CONTROLE DES ERREURS
1020 REM :
1040 REM : EMPLACEMENT 222
1060 REM :
1080 REM : ERREURS PREVUES DANS CE CAS-CI
1100 REM : 5 = FIN DES DONNEES
1120 REM : 6 = FICHIER NON TROUVE
1140 REM : 8 = ERREUR D'ENTREE/SORTIE
1150 REM :
1160 T=PEEK(222)
1180 IF T<>5 GOTO 1200
1200 REM : FIN DES DONNEES
1220 PRINT D$;"CLOSE ESSAI"
1240 PRINT "Tentative de lecture après la fin du fichier"
1250 END
1260 IF T<>6 GOTO 1290
1270 PRINT "Vérifiez qu'il s'agit de la bonne disquette"
1280 GOTO 1310
1290 IF T<>8 GOTO 1330
1300 PRINT "Vérifiez que la porte du lecteur est bien fermée"
1310 INPUT "Voulez-vous continuer ?";R$
1320 IF R$="OUI" GOTO 80
1330 PRINT "ERREUR" : END
```

EMPLOI DE L'INSTRUCTION GET

```
10 REM : PROGRAMME PROG6
20 REM
30 D$=CHR$(4)
40 PRINT D$;"OPEN BIBLIO"
50 A$=""
60 PRINT D$;"READ BIBLIO"
70 GET B$
80 IF B$="," OR B$=CHR$(13) GOTO 110
90 A$=A$+B$
100 GOTO 70
110 PRINT D$
120 PRINT A$
130 PRINT "Voulez-vous continuer ?";
140 GET R$
150 IF R$="O" GOTO 50
160 PRINT D$;"CLOSE BIBLIO"
170 END
```

sance de cause. L'emplacement du code-erreur n'est évidemment pas général et, sur d'autres ordinateurs que les Apple et compatibles, il convient de le chercher dans le manuel d'utilisation.
Avec CP/M, l'instruction GET commande le transfert de la totalité d'un enregistrement. Par contre, sur les ordinateurs Apple et compatibles (qui utilisent le DOS), GET commande la saisie d'un caractère (équivalent de INKEY$). GET peut donc saisir des données sur la disquette, caractère par caractère. Ceci peut être utile si l'on désire utiliser un séparateur de donneés non reconnu par un ordre INPUT (virgule, par exemple). Le programme ci-contre vous est proposé pour illustrer l'utilisation de GET aussi bien sur un fichier (ligne 70) qu'au clavier (ligne 140).
Les instructions du DOS concernant les fichiers à accès direct sont identiques aux précédentes à un détail près : on doit spécifier un paramètre supplémentaire correspondant à la longueur d'un enregistrement pour la commande OPEN et au numéro d'enregistrement pour la lecture et l'écriture.
Soit par exemple :

```
D$=CHR$(4)
PRINT D$;"OPEN NOM,L20"
.....
PRINT D$;"WRITE NOM,R8"
.....
PRINT D$;"READ NOM,R5"
```

Ces lignes commandent les opérations suivantes :
- Ouverture du fichier NOM dont la longueur des enregistrements est fixée à 20 caractères.
- Ecriture de l'enregistrement 8 dans ce fichier.
- Lecture de l'enregistrement 5 dans ce fichier.

Vous trouverez page 754 le listage d'un programme de lecture et d'écriture dans un fichier à accès direct.
Dans les deux types de fichiers, le DOS offre une caractéristique très intéressante : la possibilité d'accéder directement à un octet donné (du fichier en accès séquentiel ou d'un enregistrement en accès direct). Il s'agit du paramètre B.
On peut ainsi extraire le second octet de l'enregistrement 7 du fichier NOM par l'instruction :

```
PRINT D$;"READ NOM,R7,B2"
```

Le schéma de la page 755 montre les niveaux de précision de l'accès suivant les commandes employées.

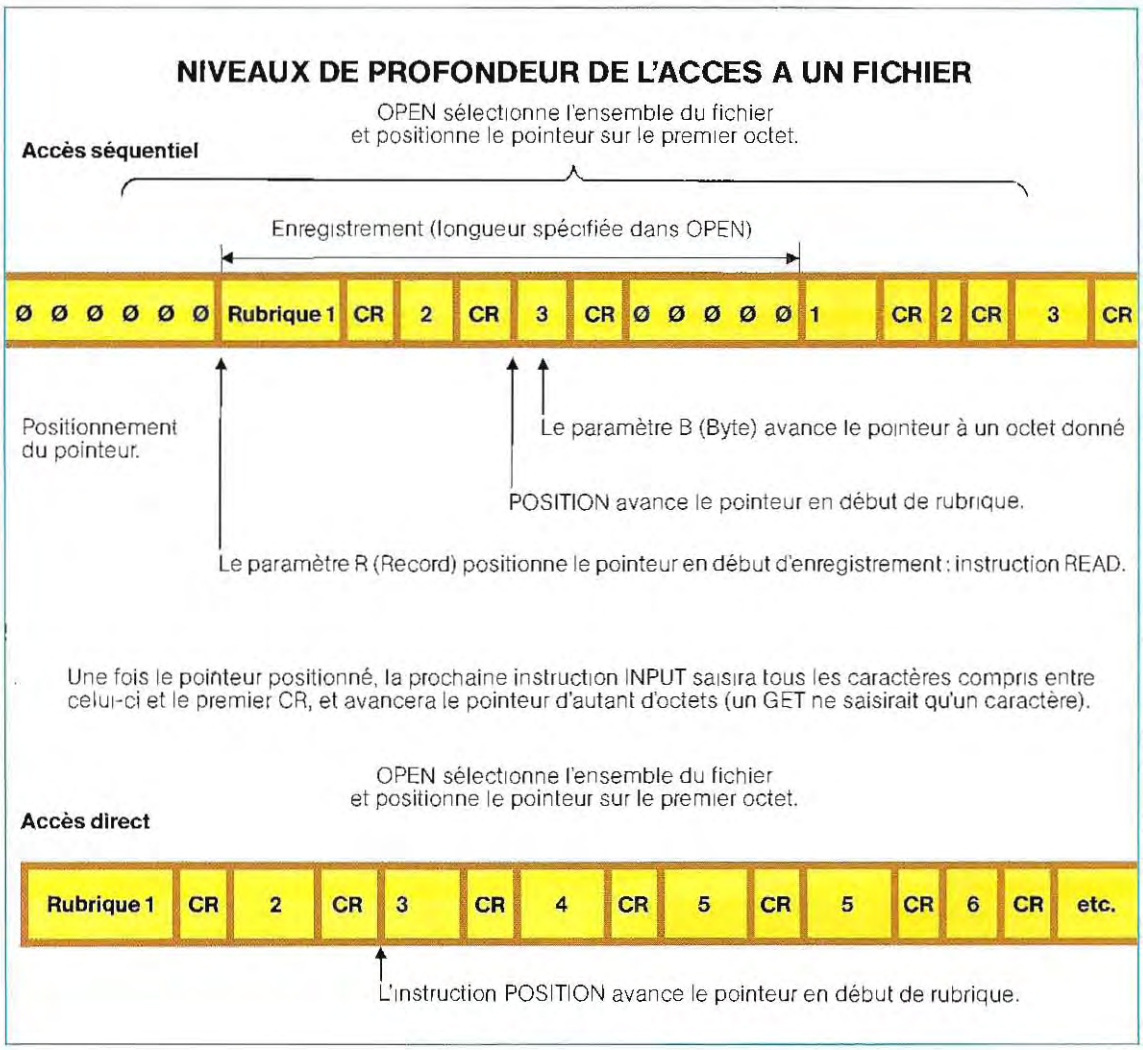
Gestion des opérations d'E/S

Les différents BASIC présentent aussi des variations importantes vis-à-vis des autres instructions d'E/S. La gestion des communications entre l'unité centrale et ses périphériques s'avère plus ou moins complexe selon les types de matériels et leur architecture interne.
En ce qui concerne la console vidéo, le mode

PROGRAMMES D'ECRITURE ET DE LECTURE EN ACCES DIRECT

```
10 REM  : PROGRAMMES PROG3
20 REM  FICHIERS ACCES DIRECT : ECRITURE
30 REM
40 D$=CHR$(4) : REM caractère contrôle du DOS
50 INPUT "Nombre de rubriques par enregistrement : ",NR
60 DIM A$(NR),A(NR),B$(NR)
70 FOR I=1 TO NR
80 PRINT "Nom rubrique";I; : INPUT A$(I)
90 INPUT "Longueur en caractères ?";A(I)
100 A(I)=A(I)+1 : REM longueur de la rubrique avec séparateur CR
110 LG=LG+A(I) : REM longueur totale d'un enregistrement
120 NEXT I
130 PRINT D$;"OPEN ESSAI3,L";LG
140 PRINT D$;"WRITE ESSAI3,R0"
150 PRINT LG : REM mémorisation de la longueur d'un enregistrement
                 dans l'enregistrement zéro
160 NE=0
170 PRINT D$;"OPEN ESSAI3,L";LG
180 NE=NE+1
190 PRINT "ENREGISTREMENT NUMERO ";NE
200 FOR I=1 TO NR
210 PRINT A$(I);" "; : INPUT B$(I)
220 IF LEN(B$(I))>A(I)-1 THEN B$(I)=LEFT$(B$(I),A(I)-1)
230 NEXT
240 PRINT "Est-ce correct ?"; : GET R$: PRINT R$
250 IF R$<>"O" GOTO 200
260 PRINT D$;"WRITE ESSAI3,R";NE
270 FOR I=1 TO NR: PRINT B$(I): NEXT
275 PRINT D$
280 PRINT "Etait-ce le dernier enregistrement ?"; : GET R$: PRINT R$
290 IF R$<>"O" GOTO 180
300 PRINT D$;"WRITE ESSAI3,R0": REM mémorisation en début de fichier
310 PRINT LG: PRINT NE: PRINT NR: REM des principaux paramètres
320 FOR I=1 TO NR: PRINT A$(I): NEXT : REM et des titres de rubriques
330 PRINT D$;"CLOSE ESSAI3"
340 END
```

```
10 REM : PROGRAMME PROG4
20 REM FICHIER ACCES DIRECT : LECTURE
30 REM
40 D$=CHR$(4)
50 PRINT D$;"OPEN ESSAI3": REM ouverture en accès séquentiel
60 PRINT D$;"READ ESSAI3"
70 INPUT LG: INPUT NE: INPUT NR
80 DIM A$(NR),B$(NR)
90 FOR I=1 TO NR: INPUT A$(I): NEXT
100 PRINT D$;"CLOSE ESSAI3"
110 PRINT D$;"OPEN ESSAI3,L";LG: REM ouverture en accès direct
120 INPUT "Quel enregistrement désirez-vous consulter ?";ENR
130 IF ENR>NE THEN PRINT "Enregistrement non affecté ": GOTO 120
140 PRINT D$;"READ ESSAI3,R";ENR
150 FOR I=1 TO NR
160 INPUT B$(I)
170 NEXT
180 PRINT D$
190 PRINT "ENREGISTREMENT ";ENR
200 FOR I=1 TO NR
210 PRINT A$(I);" : ";B$(I)
220 NEXT
230 PRINT "Voulez-vous continuer ?"; : GET R$: PRINT R$
240 IF R$="O" GOTO 120
250 PRINT D$;"CLOSE ESSAI3"
260 END
```



de liaison est à peu près standardisé autour de la norme RS 232.

Les mémoires de masse, par contre, (unités de disquettes ou disques durs) sont couplées au micro-ordinateur dans des conditions beaucoup moins homogènes et sont fortement conditionnées par l'aspect « matériel ». Ceci se traduit, comme nous l'avons vu, par un mode de gestion différent et des commandes spécifiques.

Les principales possibilités de liaison à des périphériques sont résumées sur le schéma de la page 756. Deux types d'interfaces peuvent être distingués en fonction du logiciel assurant leur contrôle :

- Interfaces gérées par le système d'exploitation.
- Interfaces programmables.

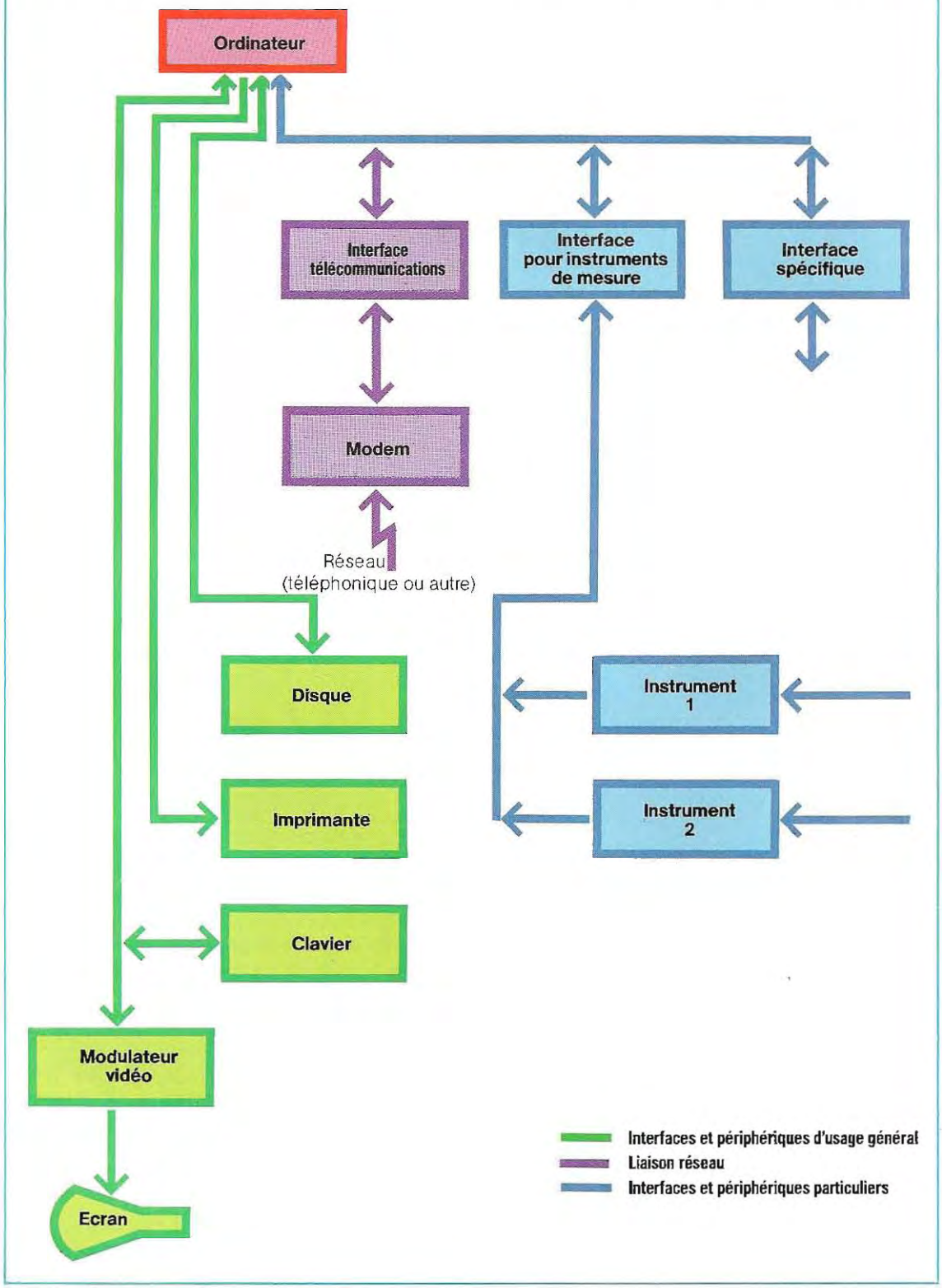
On trouve dans le premier groupe les interfaces des périphériques généralement intégrés à la configuration de base (console vidéo, imprimante, lecteur(s) de disquettes, etc.). Le langage employé comporte alors des instructions relatives à ces périphériques et leur utilisation ne nécessite pas de routines spécialisées.

Le second groupe correspond à des périphériques moins usuels pour lesquels, au contraire, n'est prévue aucune commande précise et dont la gestion implique l'écriture de routines spécialisées soit en Basic, soit en Assembleur.

Gestion des périphériques : l'interfaçage

Les ordinateurs ne sont pas aptes à commander directement toutes leurs extensions. On a

SCHEMA DE L'INTERFACE D'UN ORDINATEUR ET DE SES PERIPHERIQUES



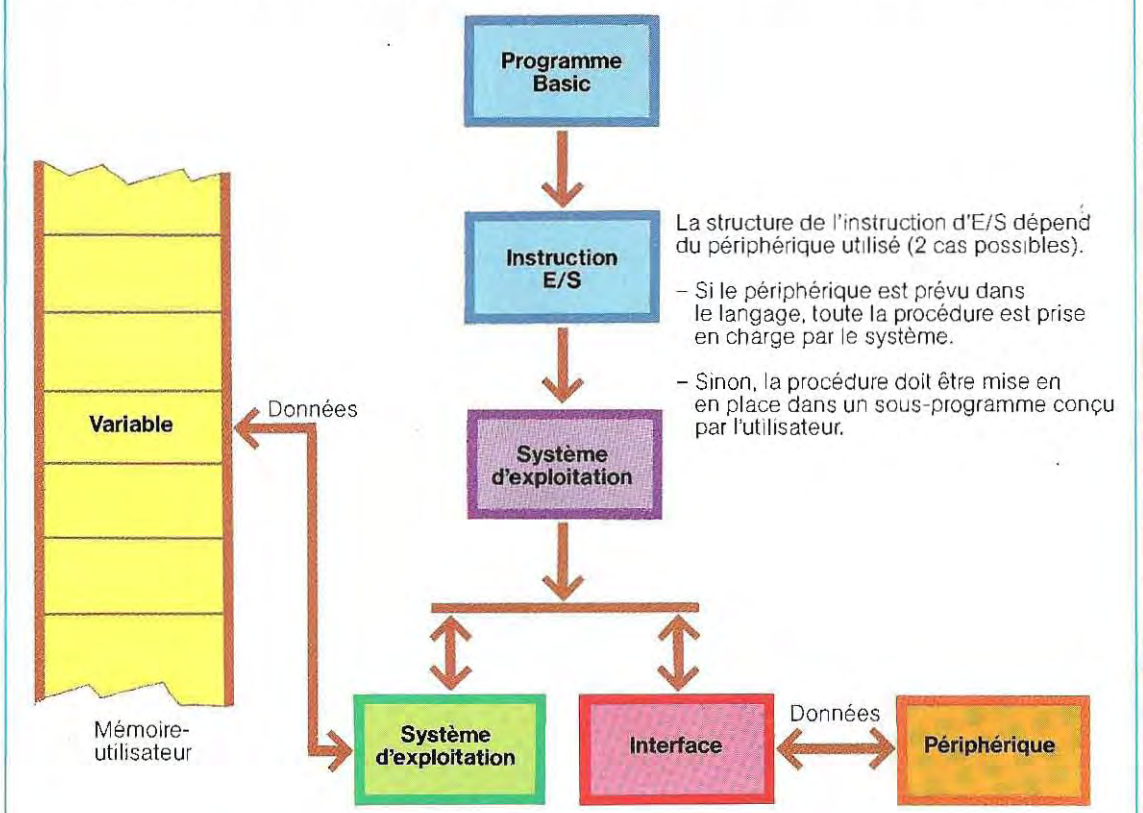
donc recours à des circuits contrôleurs couramment appelés : interfaces. Ces cartes électroniques sont branchées entre l'unité centrale et le périphérique. Elles jouent un rôle d'interprète entre ces deux unités, filtrant les signaux envoyés par l'une et/ou par l'autre, les adaptant à la logique du destinataire et synchronisant l'ensemble de ces échanges. On peut donc résumer ainsi les fonctions d'une carte d'interface :

- Adaptation du niveau des signaux.
- Mise des données au format.
- Temporisation des signaux.

Il arrive fréquemment que les signaux logiques (potentiels électriques correspondant aux états 1 et 0) d'une ligne donnée soient interprétés de façon inversée par l'ordinateur et le périphérique. C'est alors l'interface qui transforme les signaux de façon appropriée afin qu'ils puissent être reconnus par le destinataire.

De même, la transmission des signaux s'effectue en parallèle sur la plupart des micro-ordinateurs. Par contre, certains périphériques (modem, par exemple) fonctionnent avec des signaux transmis en série. C'est encore à l'interface qu'incombe la transformation parallèle-série et inversement. Enfin, les périphériques comportant souvent des pièces mécaniques en mouvement travaillent beaucoup moins vite que l'ordinateur. L'interface doit donc réguler le flot des données. Une des méthodes employée pour résoudre ce type de problèmes est la procédure dite de « handshaking ». Les données sont transmises au coup par coup : on attend un accusé de réception du destinataire pour envoyer la suite si tout s'est bien passé. Le mécanisme de l'échange des données est illustré par le schéma ci-dessous. Le programme d'application sera plus ou moins complexe selon le type de périphérique utilisé.
Si le langage disponible prévoit des instruc-

MECANISME DE L'ECHANGE DES DONNEES AVEC UN PERIPHERIQUE



tions pour la gestion de la périphérique, tous les paramètres nécessaires à la procédure d'E/S seront mis en place par le système d'exploitation. Il suffira au programmeur d'employer la ou les instructions appropriées (INPUT, PRINT ou PRINT USING, par exemple). Leur décodage appelera automatiquement les programmes système spécifiques à ce périphérique.

Si, au contraire, le langage ne possède pas d'instructions relatives au périphérique voulu, l'écriture des modules de gestion incombera au programmeur.

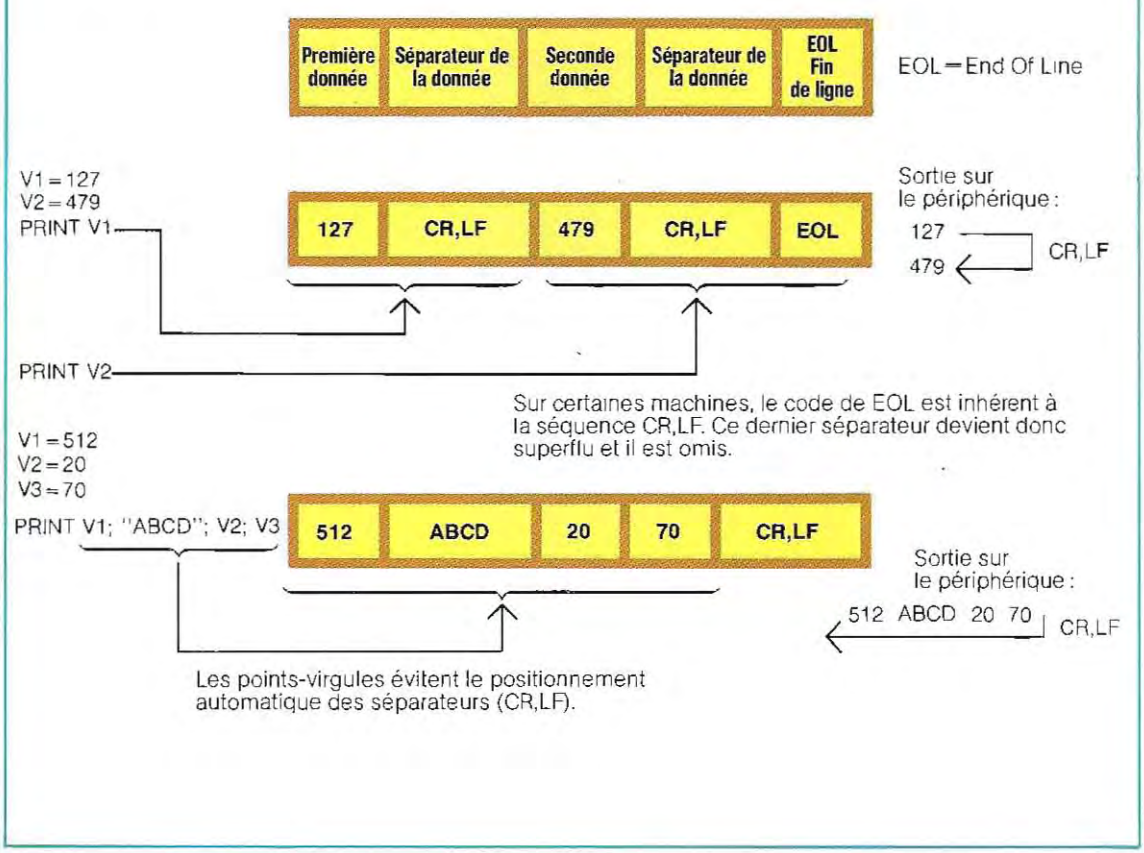
La réalisation de ces routines dépendra alors essentiellement du matériel. En effet, certaines interfaces peuvent être commandées par des programmes en Basic. Dans le cas contraire, la routine devra être écrite en Assembleur. Deux exemples de transmission vers des périphériques de sortie (l'écran et l'imprimante, par exemple) sont présentés ci-dessous. L'instruction PRINT suffit à activer le mécanisme de sortie et c'est le système qui se charge d'ajouter les divers séparateurs et caractères de contrôle (par exemple, CR = Retour chariot ou LF = Ligne suivante). Est également pris en compte lors de l'exécution de la commande, un éventuel format de sortie. Le schéma du haut de cette page illustre deux emplois de l'instruction PRINT USING avec le symbole du Basic 80: #. D'autres versions du Basic offrent le choix entre plusieurs symboles, suivant le format désiré. Voici les principales options :

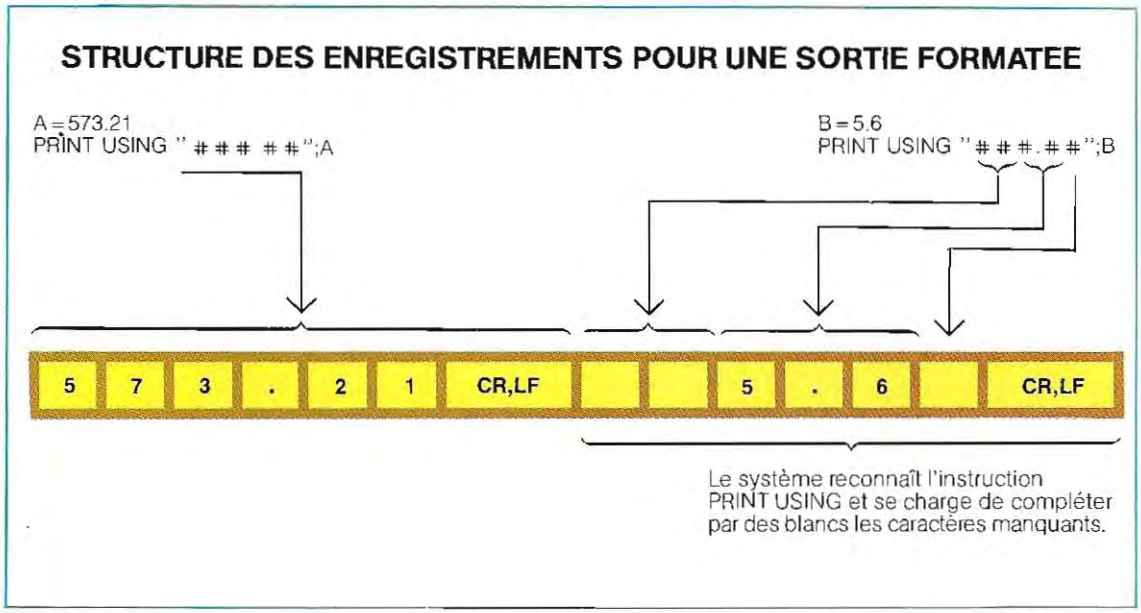
A représente un caractère alphanumérique. Ainsi, USING "4A" prévoit la sortie de 4 caractères ASCII.

D abréviation de Digit (= chiffre). Chaque D correspond à un chiffre décimal (de 0 à 9). L'expression numérique sera précédée de son signe si on mentionne l'option S. Soit :

USING "SDD.D"

STRUCTURE DES ENREGISTREMENTS EN SORTIE SUR UN PERIPHERIQUE





Cette instruction demande un nombre signé et composé de deux entiers et d'une décimale.

E signifie exposant. Le nombre sera représenté en notation exponentielle.

K désigne le format condensé. Tous les blancs seront supprimés.

X commande l'insertion d'un certain nombre d'espaces blancs. Soit l'instruction :

```
USING "SD.DD,5X,SDD.D"
```

Cinq espaces (5X) seront imprimés entre le premier nombre (SD.DD) et le second (SDD.D).

Nous avons volontairement omis, dans ces exemples, l'instruction de sortie dont USING doit obligatoirement être précédé. En effet, si PRINT est le plus souvent employé, d'autres commandes comme OUTPUT sont en vigueur sur certaines machines. Voici donc un exemple d'instruction complète :

```
R=-51.6
OUTPUT 3 USING "SDDD.DD";R
```

Celle-ci génère sur le périphérique N° 3 une donnée constituée d'un signe (S), de trois chiffres (DD), d'un point (.) et de deux décimales (D) :

```
-051.60
```

Nous avons ici le cas d'un ordinateur où le périphérique de sortie est désigné par son numéro. Ce numéro est celui du port (slot en anglais) dans lequel est insérée l'interface du périphérique concerné. Il peut être introduit dans chaque instruction de sortie (comme dans cet exemple) ou fixé préalablement à plusieurs sorties (ou entrées) : PR#n (ou IN#n) sur Apple. Cette notation évite le recours à des instructions spécifiques pour l'écran, l'imprimante, les disquettes, etc.

Le terme de port désigne de longs connecteurs, généralement reliés au bus de l'ordinateur, et dans lesquels peuvent être insérées non seulement des cartes d'interfaces mais aussi d'autres éléments tels que des extensions de mémoire.

Sur les ordinateurs utilisant ce système, le clavier et l'écran sont les organes d'E/S implicites. On change donc de périphérique en activant le port auquel il est relié. Très répandu sur les ordinateurs individuels, ce mode de gestion des périphériques présente plusieurs avantages et offre notamment à chacun la possibilité d'adapter la configuration du système à ses besoins. Ce système, en effet, facilite considérablement l'adjonction d'un nouveau périphérique : on le connecte à l'un des ports et il n'y aura qu'à préciser le numéro de celui-ci pour qu'il vienne supplanter l'écran et/ou le clavier.

En outre, il est possible de compléter l'installation par un second microprocesseur. Cela

est très important car les systèmes d'exploitation sont développés pour un microprocesseur donné. Ainsi, le CP/M et ses différentes versions sont prévus pour les microprocesseurs Z80, 8080, 8086,... alors que d'autres systèmes comme le DOS sont conçus pour des microprocesseurs différents.
L'adjonction d'une seconde unité de traitement permet donc de disposer de deux microprocesseurs et de leurs deux systèmes d'exploitation associés.
Outre l'impression de posséder deux ordinateurs, une telle configuration permet de choisir le système le mieux adapté pour chaque application et surtout, ouvre les portes des deux plus importantes bibliothèques de progiciels dans le cas de l'association DOS+CP/M. A titre d'exemple, voici deux séquences d'instruction demandant une sortie sur l'écran, puis sur l'imprimante, l'une sous CP/M et l'autre sous DOS.

```
CP/M
10  A$="ESSAI"
20  PRINT A$    :' AFFICHAGE
30  LPRINT A$   :' IMPRESSION

DOS
10  A$="ESSAI"
20  PRINT A$    : REM  AFFICHAGE
30  PRINT CHR$(4);"PR#1"   : REM
                IMPRIMANTE EN LIGNE
40  PRINT A$    : REM  IMPRESSION
```

La ligne 30 sert, dans les deux cas, à diriger la sortie vers l'imprimante.
Sous DOS, CHR$(4) est le code CTRL-D précède la commande et la chaîne PR#1 est la commande proprement dite.
Cette procédure est commune aux nombreux ordinateurs bâtis autour du microprocesseur 6502 (APPLE, SIPREL, etc.) constituant aujourd'hui la plus grande famille de machines ne travaillant pas sous CP/M.
La gestion de l'écran présente également quelques particularités.
Les Basic conçus pour le CP/M ne possèdent généralement pas d'instructions de positionnement du curseur. Celui-ci est déplacé sur l'écran par l'intermédiaire de codes de contrôle dont l'ordre et la valeur dépendent de l'unité vidéo. En revanche, dans la famille des ordinateurs à microprocesseur 6502, il existe deux instructions Basic de positionnement du curseur :

HTAB n pour le positionnement horizontal
VTAB m pour le positionnement vertical

Les lignes suivantes montrent comment on peut les utiliser :

```
10  INPUT "Colonne";C
20  INPUT "Ligne";L
30  A$="Essai de positionnement"
100  HTAB C:VTAB L
110  PRINT A$
120  END
```

Contrairement à TAB(X), ces instructions s'utilisent donc en dehors d'une instruction PRINT.

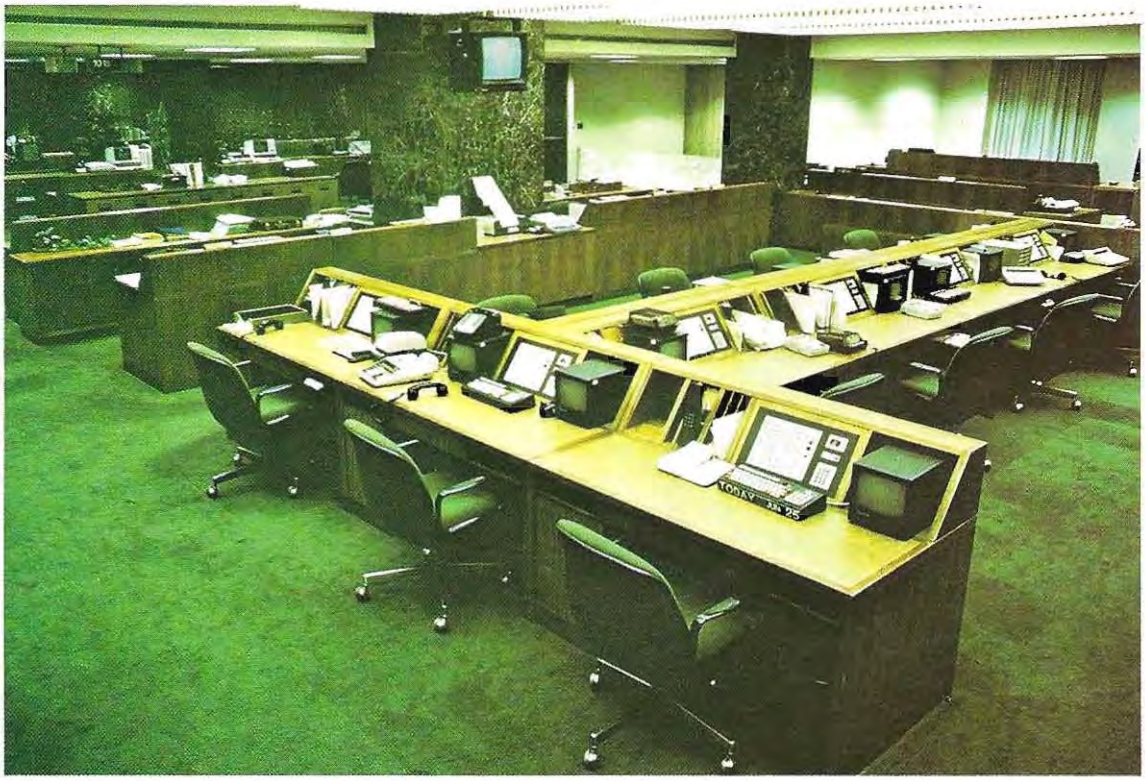
Transmission et réception des données

Le besoin croissant d'obtenir des informations en temps réel est à l'origine du développement des systèmes d'informatique répartie, c'est-à-dire de réseaux constitués de plusieurs micro-ordinateurs reliés entre eux ou à un gros ordinateur. Par rapport à de simples terminaux, les micro-ordinateurs offrent l'avantage d'une puissance de calcul indépendante qui leur permet d'assurer à la fois un rôle de terminal et d'unité de traitement autonome.
Le protocole de transmission des données le plus répandu est l'EIA RS-232-C. C'est ainsi que certains constructeurs équipent leurs machines d'interfaces répondant à ce standard et d'instructions Basic permettant de contrôler le protocole de transmission. L'instruction la plus importante est celle qui fournit les paramètres correspondant aux modalités de cette liaison, soit :

– **La vitesse de transmission**, exprimée en bauds (bits par seconde). Les différentes vitesses pratiquées sont : 75, 110, 150, 300, 600, 1200, 1800, 2400, 4800, 9600 bps.

– **La parité**. Elle indique le type du contrôle de parité, c'est-à-dire la façon dont est traité le bit de parité. Il y a cinq possibilités :

— non transmis
— toujours 0



K. Reese/Marka

Dans les agences de courtage, informatisation est synonyme de rapidité, et donc d'efficacité.

— toujours 1
— transmis pour la parité impaire
— transmis pour la parité paire

– **Bits de données**. Les données peuvent être symbolisées par un nombre de bits compris entre 4 et 8. Il faut donc configurer l'interface pour le format choisi.

– **Bits d'arrêt**. Des bits d'arrêt peuvent être ou non positionnés à la suite de chaque donnée. Il y a trois possibilités :

— pas de bit d'arrêt
— un seul bit d'arrêt
— deux bits d'arrêt

– **Numéro du fichier**. Pour utiliser un canal de télécommunications, on doit lui allouer un numéro de fichier de façon à pouvoir ensuite employer les instructions GET et PUT puisqu'elles agissent sur des fichiers.
Considérons l'instruction suivante :

```
OPEN "COM 1:2400,S,8,1"AS#3
```

Elle commande l'ouverture d'un fichier de télécommunication portant le numéro 3. Les sorties s'effectueront par l'interface 1 (COM 1) à une vitesse de 2400 bps, selon un format de 8 bits plus 1 bit d'arrêt et sans bit de parité (symbole S).
On peut également indiquer dans cette instruction les codes de contrôle des signaux de ligne. D'utilisation très simple, le RS-232-C est le protocole le plus adapté à la liaison de deux unités de calcul. En revanche, dès que ce nombre augmente, le RS-232-C nécessite la multiplication des ports d'accès ou un système de sélection extérieur. Une telle installation devient vite très complexe et remet en question le choix du protocole d'échange.
Or, des systèmes comme le contrôle automatique d'appareils de mesure requièrent l'échange simultané de données entre une unité de traitement et un certain nombre de périphériques. Ces applications ont donc suscité la conception et la mise au point de protocoles spéciaux dont le plus connu, l'HP-IB (Hewlett-Packard Interface Bus), a été proposé à l'IEC (International Electrotechnical Commission) comme standard international.

Le bus HP-IB comporte 16 câbles dont huit sont réservés aux données et les autres à des signaux de contrôle. Les données sont transmises en ASCII sur 7 bits (le huitième bit étant le bit de parité). Les appareils et, d'une manière générale, les unités à connecter sont répartis en trois catégories :

Les RECEPTEURS (listener) sont des unités capables comme l'écran ou l'imprimante de recevoir des données provenant d'autres unités.

Les EMETTEURS (talker) envoient, au contraire, des données vers l'extérieur. Ce sont des enregistreurs, ou tout appareil effectuant des mesures et les convertissant en signaux électriques (numériques) avant de les acheminer vers le bus.

Classement des listages et des états dans la salle des imprimantes. Des sigles permettent d'identifier l'utilisateur.

J. Pickerell/Marka

Les CONTROLEURS (controller) sont des unités qui gèrent et contrôlent les échanges entre les autres unités. Ce sont généralement des ordinateurs.

Certaines unités peuvent appartenir successivement à l'une ou l'autre de ces catégories : un instrument de mesure est RECEPTEUR lorsqu'il reçoit les instructions de l'ordinateur et EMETTEUR lorsqu'il communique les résultats de ses mesures. L'HP-IB est conçu pour qu'il n'y ait, à un instant donné, qu'un EMETTEUR et qu'un CONTROLEUR qui fonctionnent en tant que tels. Les autres appareils ne peuvent, à cet instant, jouer que le rôle de RECEPTEURS.

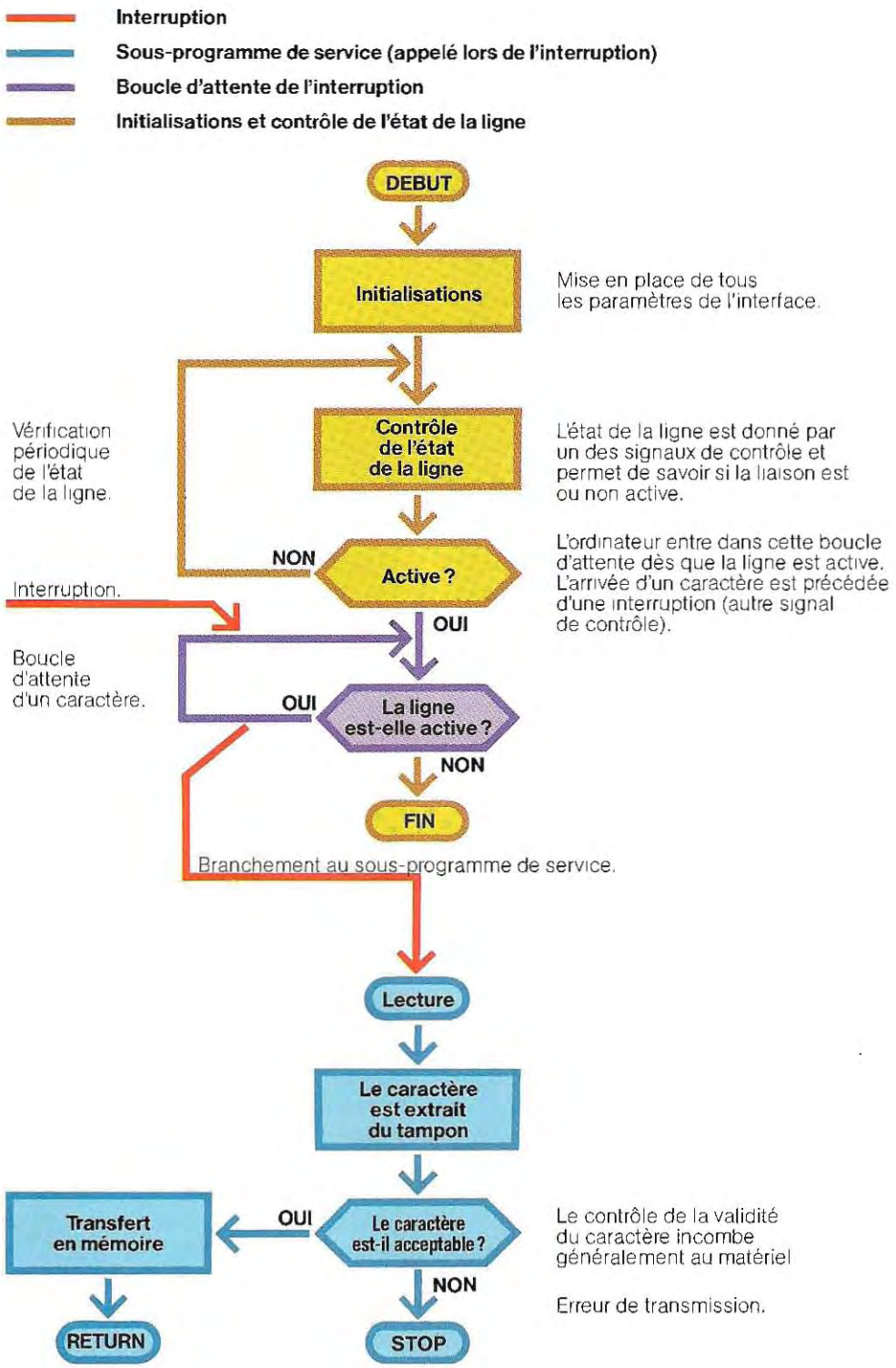
Certaines versions du Basic possèdent des instructions de piégeage des données acheminées par le bus. L'arrivée de ces données provoque l'interruption de l'opération en cours et l'appel d'un sous-programme de service conçu pour leur acquisition. Les mécanismes commandés par ces instructions permettent à l'ordinateur de poursuivre sa tâche entre la réception de deux caractères consécutifs, ne s'interrompant que pour la saisie de ceux-ci. L'organigramme de la page 763 illustre le principe de la réception des données transmises par le bus.

Déclenchée par un signal d'origine externe, l'interruption (en anglais, trap ou interrupt) est une opération primordiale pour l'acquisition des données car c'est elle qui déclenche l'appel des sous-programmes appropriés. Prenons l'exemple de la commande RESET qui est activée en CP/M par les codes CTRL-C et interrompt l'exécution de n'importe quel programme. Pendant le déroulement d'une application, l'ordinateur n'interroge cependant pas le clavier pour y acquérir d'éventuelles commandes. Ce type d'entrée n'intervient donc pas comme un caractère normal mais agit directement sur la ligne des interruptions. Le traitement en cours est alors suspendu et on peut éventuellement effectuer un branchement sur une opération particulière.

Sur certaines machines, on dispose de touches (key) ou de molettes (knob) pour provoquer une interruption.

Ces touches d'interruption sont des touches de fonction qui engendrent des interruptions

EXEMPLE DE PROGRAMME DE TELECOMMUNICATION



diverses donnant chacune lieu à l'exécution d'un sous-programme déterminé. Il existe donc dans le Basic de ces machines des instructions permettant au programmeur de définir les opérations associées à la frappe de chaque touche. Ainsi, l'instruction :

ON KEY 3 GOTO 100

détermine un branchement à la ligne 100 après l'interruption provoquée par la frappe de la troisième touche spéciale. Quant aux molettes, elles ont généralement une fonction de déplacement du curseur. Il s'agit d'un dispositif dont la rotation génère des impulsions. L'ordinateur calcule l'angle de rotation en fonction du nombre d'impulsions reçues et déplace le curseur en conséquence. Il existe également en Basic une instruction (ON KNOB) qui permet de sauter à l'endroit voulu d'un programme par la manipulation d'une molette au lieu d'une touche. La gestion des interruptions n'est toutefois pas aussi simple qu'on pourrait le croire, en raison notamment de la plus ou moins grande priorité des périphériques.

Nombreux sont les périphériques qui peuvent demander eux-mêmes une interruption. Aussi est-il tout à fait possible que deux demandes parviennent presque simultanément à l'ordinateur ou encore qu'une demande soit transmise en cours d'exécution d'une routine, elle-même appelée par une autre interruption. C'est pourquoi une priorité est accordée à chaque périphérique afin de permettre au système de décider s'il doit accepter l'interruption ou mettre la demande sur une « liste d'attente ». Le périphérique ayant la priorité la plus élevée provoquera une interruption immédiate alors que les autres demandes seront retardées, voire ignorées.

La priorité absolue d'une opération est déterminée soit par le matériel, soit par le logiciel. Ce dernier cas est réservé au traitement d'un incident ou d'une erreur et on le programme grâce à des instructions spéciales en suivant la logique exposée ci-dessous.

Si aucune interruption n'est en cours et que le programme s'exécute normalement, le degré de priorité est fixé à 0 (on appelle ce degré « priorité système » car il est attribué automatiquement par le système). Toute interruption pourra donc être prise en compte immédiatement – ou, plus exactement, au terme de l'instruction en cours d'exécution – puisque, quel que soit son degré de priorité, il ne pourra, en aucun cas, être inférieur à zéro.

Si une nouvelle demande d'interruption survient pendant l'exécution de la routine appelée par la première, deux cas peuvent se présenter :

1 / soit la nouvelle interruption a une priorité inférieure à la précédente et elle est mise en attente jusqu'au terme de celle-ci ;

2 / soit son degré de priorité est supérieur ou égal à celui de l'interruption en cours. Elle est alors prise en compte immédiatement. Le programme appelé par la première s'interrompt et ne reprendra qu'ensuite.

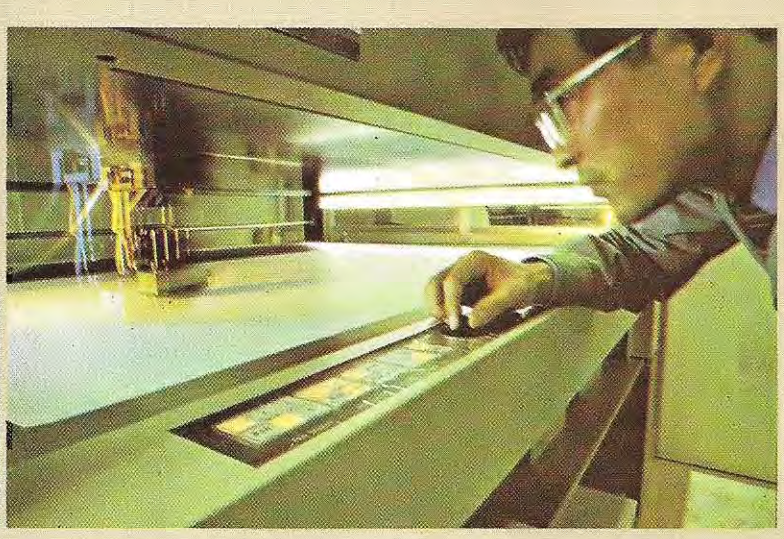
La priorité logicielle détermine l'ordre de prise en compte des interruptions dont la demande a été reçue, alors que la priorité « hardware » détermine l'ordre de réception des demandes. En d'autres termes, elle sert à planifier les demandes d'interruption en provenance des périphériques. Ces deux priorités sont indépendantes.

Lorsque l'ordinateur reçoit une demande d'interruption, il abandonne la tâche en cours d'exécution et recherche l'interface d'où émane cette demande.

Puis, il met cette demande sur une liste d'attente, négligeant désormais les autres demandes de cette interface. Après avoir exécuté toutes les opérations prioritaires, il examine à nouveau les demandes non satisfaites.

La gestion des interruptions est capitale dans les transmissions et ce, autant en ce qui concerne l'acquisition des données que pour le contrôle des durées d'exécution. En effet, le matériel n'est jamais totalement à l'abri d'un incident pendant la transmission des données. Une détérioration de la ligne ou, plus fréquemment, une micro-coupure de l'alimentation peuvent être à l'origine d'un défaut de transmission. Il est donc indispensable de prévoir un contrôle des temps de transmission pour éviter que l'ordinateur attende indéfiniment une donnée. Il existe donc une interruption, conditionnée par la durée de l'attente, qui provoque une sortie automatique de la boucle d'attente. Un message de diagnostic est émis et l'ordinateur passe à la tâche suivante.

Du silicium à l'ordinateur



M. Wolf/Black Star-Grazia Neri

Les ingénieurs font appel à toutes les ressources de la C.A.O. (Conception Assistée par Ordinateur) pour développer les nouveaux circuits intégrés.

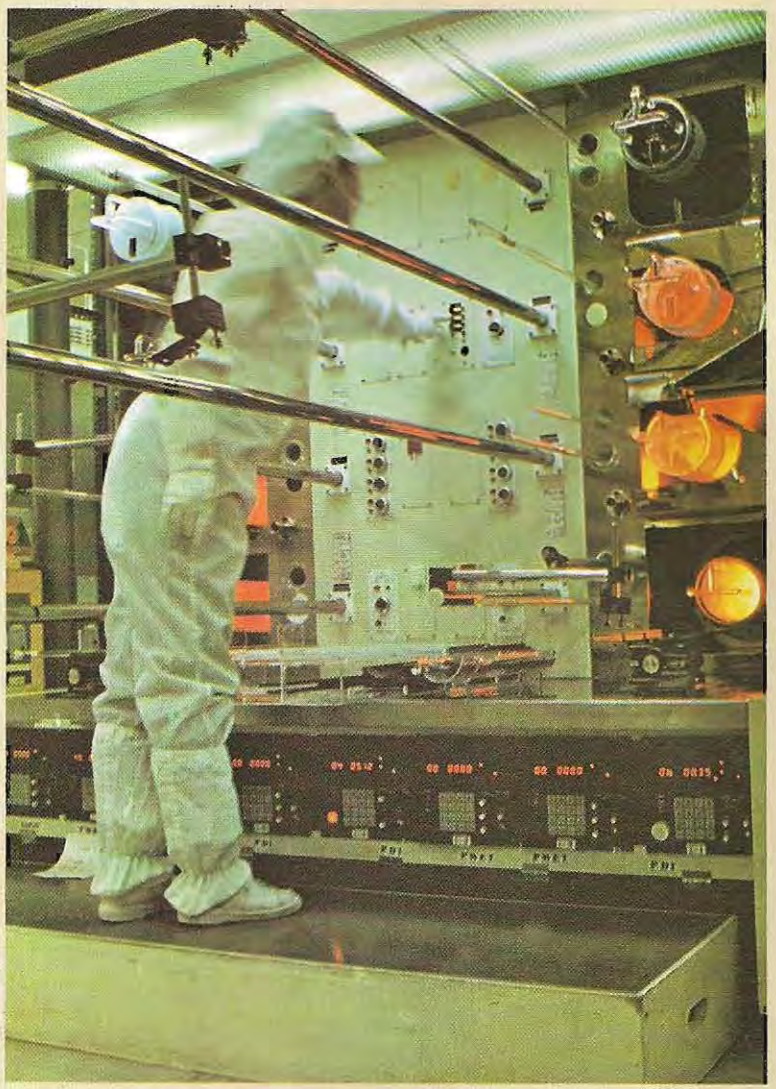
La photographie du haut a été prise pendant le dessin des motifs d'un circuit à l'aide d'un traceur à haute résolution. Le concepteur peut obtenir la visualisation sur un écran de la structure du circuit mémorisée par l'ordinateur, effectuer des modifications au clavier jusqu'à la mise au point définitive et, enfin, réaliser les masques nécessaires au procédé de photolithographie.

Stock Photos/Grazia Neri

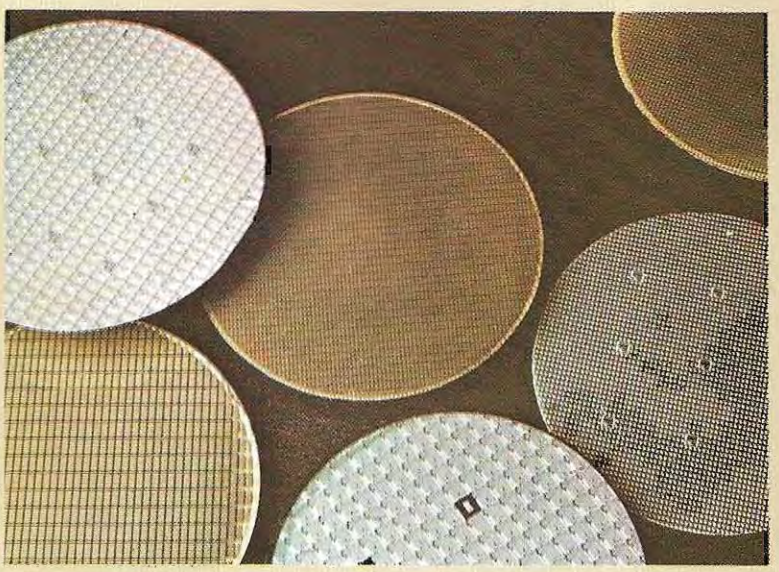
On additionne des impuretés au silicium très pur en fusion afin de lui conférer les propriétés électriques désirées. Les atomes de silicium viennent ensuite se ranger autour d'un minuscule noyau, donnant naissance à un monocristal presque parfait de silicium semi-conducteur. On extrait alors lentement (seconde photographie) le barreau de silicium auquel on donne un diamètre régulier avant de le découper transversalement en fins disques circulaires, les «tranches» (wafers, en anglais). Après avoir soigneusement poli ces tranches, on y imprime les différentes couches du circuit par photolithographie.

M. Wolf/Black Star-Grazia Neri

La photographie du bas illustre le bombardement des zones conductrices par de l'oxygène fortement ionisé.



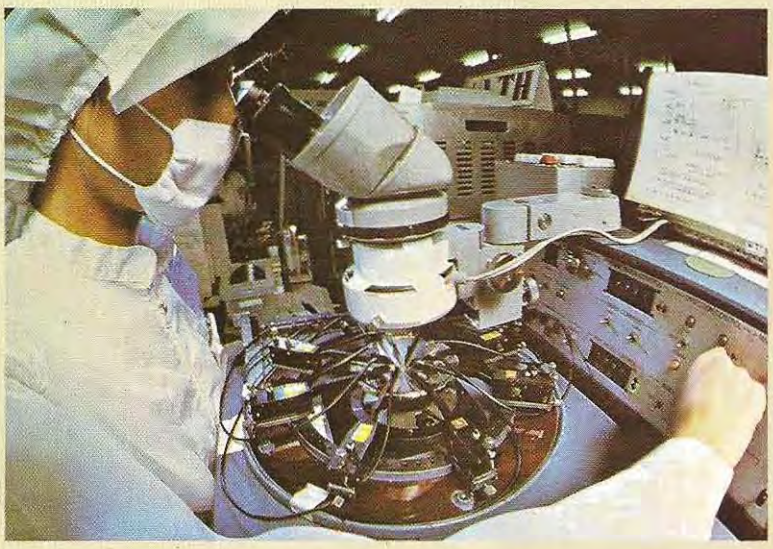
M. Wolf/Black Star-Grazia Neri



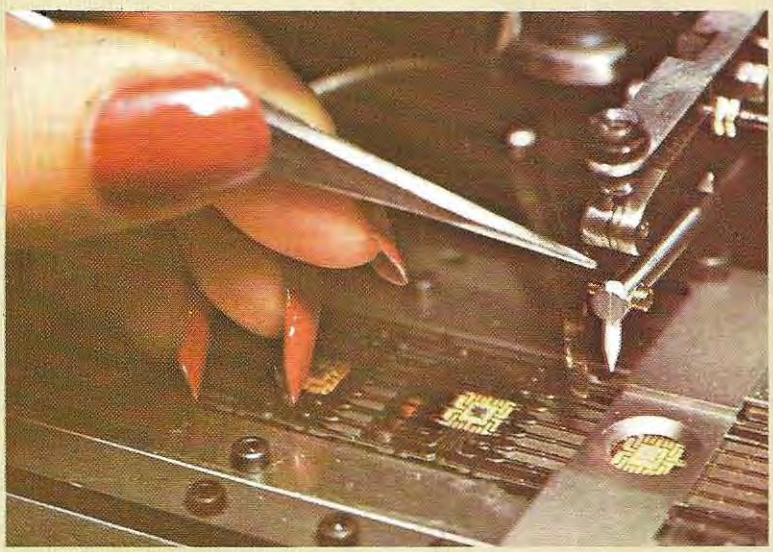
B. Coleman/Marka

Les tranches de silicium doivent être traitées dans une atmosphère parfaitement contrôlée. Aussi les scelle-t-on avec des impuretés dans des ampoules de quartz qui sont ensuite placées dans des fours à haute température (photographie du haut) où les atomes des impuretés parvenues à l'état gazeux diffusent dans le silicium à travers les «fenêtres» créées par photolithographie: c'est le dopage.
On obtient ainsi les transistors, les résistances et les diodes constitutifs du circuit intégré.
A l'issue de la gravure, les tranches se présentent comme sur la photo du bas. On voit d'ailleurs nettement qu'en raison de la forme circulaire des tranches, les circuits qui se trouvent sur leur pourtour sont incomplets. Inutilisables, ils seront mis au rebut lors du découpage. Les puces coûtent d'autant moins cher que le circuit est plus petit, car on peut en imprimer davantage sur chaque plaquette. Avant le découpage, chaque tranche est examinée au microscope et les circuits défectueux sont marqués au vernis coloré.
Un mauvais fonctionnement des puces peut avoir des origines très différentes. Du fait de l'extrême miniaturisation, le moindre grain de poussière, invisible à l'œil nu, peut couper une connexion ou même plusieurs.
C'est pourquoi on apporte un soin rigoureux à toutes les étapes de la fabrication. On prend le maximum de précautions pour éviter la contamination des circuits par des corps étrangers. Une équipe du laboratoire

M. Wolf/Black Star-Grazia Neri

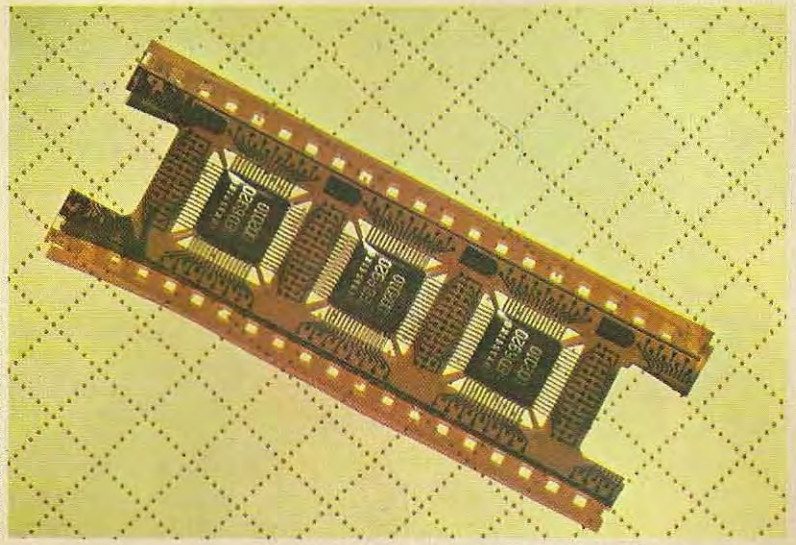


M. Wolf/Black Star-Grazia Neri

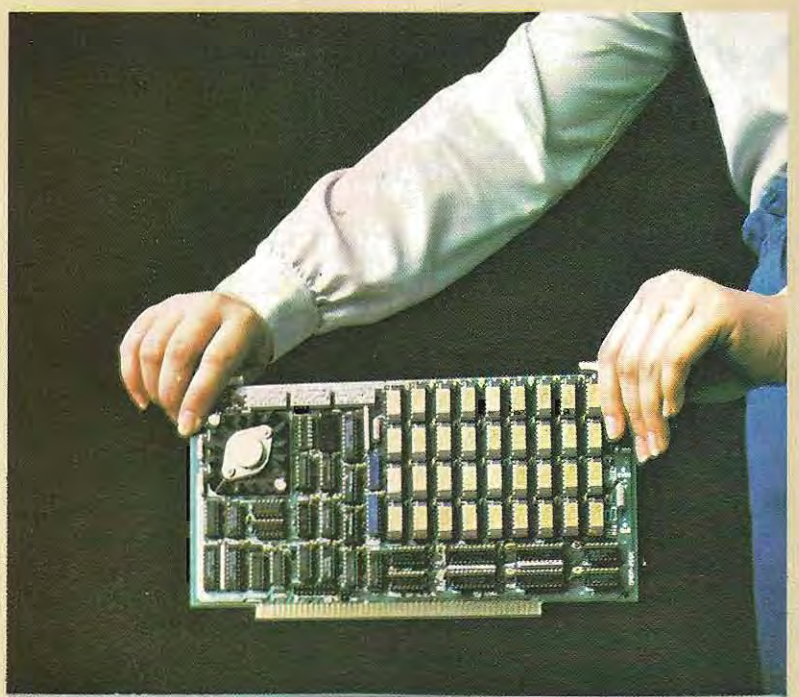


K. Reese/Marka

IBM new yorkais de East Fishkill a récemment développé une technique expérimentale de contrôle qui devrait permettre une nette amélioration du rendement de la production des puces : elle ne comprend pas moins de 300 phases.
La réalisation d'un transistor ou d'un autre composant consiste à déposer des impuretés, ou dopants, sur une tranche de silicium.
Ces dopants sont les éléments caractéristiques du composant. Ils doivent être disposés à des endroits déterminés avec une précision de l'ordre de l'atome.
Sinon, le circuit est défectueux. Le dépôt de la substance dopante suppose donc des mesures extrêmement minutieuses.
Cette nouvelle méthode de contrôle garantit le positionnement parfait des masques et donc, la disposition correcte des circuits sur la microplaquette.
Le positionnement des masques est, en effet, l'une des étapes les plus délicates.
Il s'agit de disposer, sur des puces d'une quarantaine de millimètres carrés, des milliers de composants dont la taille est d'environ un micron (l'épaisseur moyenne d'un cheveu est de 250 microns).
Après cet examen optique, on teste le fonctionnement des circuits encore réunis sur les tranches (photo du milieu).
Chaque circuit est soumis au contrôle d'un testeur commandé par ordinateur. Des pointes conductrices, placées automatiquement aux endroits voulus, transmettent



M. Wolf/Black Star-Grazia Neri



B. Coleman/Marka

des signaux électriques de faible intensité. La réponse du circuit intégré est alors comparée à une réponse-type et la plus légère différence entraîne son élimination.
Lorsque tous les contrôles ont été effectués, on procède au découpage des plaquettes à l'aide d'une lame diamantée commandée par une machine micrométrique. Les « mauvais » circuits sont rejetés tandis que les « bonnes » puces sont câblées dans une machine spéciale afin de pouvoir être raccordées à l'extérieur (photo du bas de la page 768).
Une fois tous les fils de connexion soudés, les puces sont encapsulées dans des supports de céramique qui les protègent et les isolent définitivement. Seuls dépassent encore les fils de connexion.
Comme ils sont également très fragiles, on les protège à leur tour en insérant les éléments de céramique à la suite les uns des autres dans une bande qui rappelle une pellicule photographique (photo du haut).
Les perforations qui se trouvent de chaque côté permettent de manipuler les composants sans les endommager, notamment pour leur montage et leur soudage sur les plaques.
On peut voir sur la photographie ci-contre une plaque de composants prête à être placée dans le logement qui lui est destiné sur une machine.
L'insertion suffit généralement à établir les connexions, même s'il est parfois nécessaire de réaliser quelques soudures pour les rendre plus fiables.

Le logiciel

Un logiciel d'application est destiné à résoudre un problème déterminé. Progiciel standardisé ou programme « sur mesure », il doit :

- permettre la réalisation des traitements désirés et l'obtention des sorties voulues ;
- présenter une fiabilité satisfaisante.

La première de ces conditions est claire et impérative. Si un programme ne répond pas aux besoins exprimés; il n'y a plus qu'à le modifier ou, éventuellement, à revoir ses spécifications.
Il est simple de s'assurer de la correspondance entre prestations et spécifications : on fournit en entrée des données précises et l'on doit obtenir en sortie les résultats prévus.
Il s'avère beaucoup plus délicat de déterminer si la seconde condition est remplie, et la réponse n'est pas aussi catégorique. La démonstration de la justesse d'un programme n'est, en effet, jamais définie, car une erreur peut très bien échapper aux contrôles les plus rigoureux si elle ne se manifeste que sous certaines conditions particulières.
Ces erreurs masquées connaissent parfois des répercussions considérables. Ainsi, celles qui subsistaient dans les programmes du projet spatial Apollo – malgré les 600 millions de dollars que coûta leur développement ! – furent à l'origine de la plupart des difficultés qui entachèrent ce projet, et une erreur de ce type alla même jusqu'à entraîner l'échec de la première mission d'une sonde Vénus.
Toutefois, les logiciels sont loin d'atteindre une telle complexité dans les applications courantes et le risque de commettre des erreurs demeure beaucoup plus faible. De surcroît, on réussit généralement a les déceler et à les expurger.
Pour obtenir un programme sans erreur, il faut conjuguer deux actions complémentaires : la prévention et la correction.
La prévention commence dès la phase de conception du logiciel, et conditionne toute la suite de son développement. Quant à la correction, elle n'intervient, bien sûr, qu'en dernier

Une vue impressionnante des installations d'une salle de commandes.

Italcable

lieu, lors du contrôle et de la validation, mais elle est également déterminée par la façon dont le travail a été abordé.
Pour prévenir les erreurs, on peut développer le projet de façon à faire ressortir les modules du logiciel présentant les plus grandes difficultés. Il existe, pour chaque type de programme et pour chaque étape, des méthodes de conception et de programmation rigoureuses et complexes. C'est pourquoi nous ne traiterons que les techniques fondamentales, sélectionnant les aspects les plus importants pour les machines de la catégorie qui nous intéresse. Les points essentiels de la réalisation d'un logiciel d'application sont les suivants :

- définition des objectifs ;
- architecture du système, et, notamment, caractéristiques des interfaces ;
- structure du logiciel ;
- méthodes de programmation et choix des langages.

Il faut, avant tout, parvenir à une vision d'ensemble des buts à atteindre.
On distingue deux types d'objectifs : à côté des besoins spécifiques de l'utilisateur (qui constitueront l'essentiel des traitements), il existe un certain nombre d'objectifs à caractère général, que le programmeur ne doit jamais perdre de vue.
Tout bon logiciel d'une certaine ampleur doit en effet présenter, à un degré variable, les qualités suivantes :

Généralisation. Le développement d'un logiciel suppose toujours un lourd investissement. Aussi, un projet très spécifique, c'est-à-dire n'intéressant que peu d'utilisateurs, s'avère généralement peu rentable. Toutefois, il ne faut pas tomber dans l'excès inverse, car toute généralisation entraîne l'allongement du programme et les possibilités d'erreur s'en trouvent multipliées.
C'est au concepteur de savoir évaluer, cas par cas, si telle ou telle généralisation s'impose vraiment, et sera facile à mettre en œuvre sans que les délais d'apprentissage deviennent une charge trop lourde à l'utilisateur.

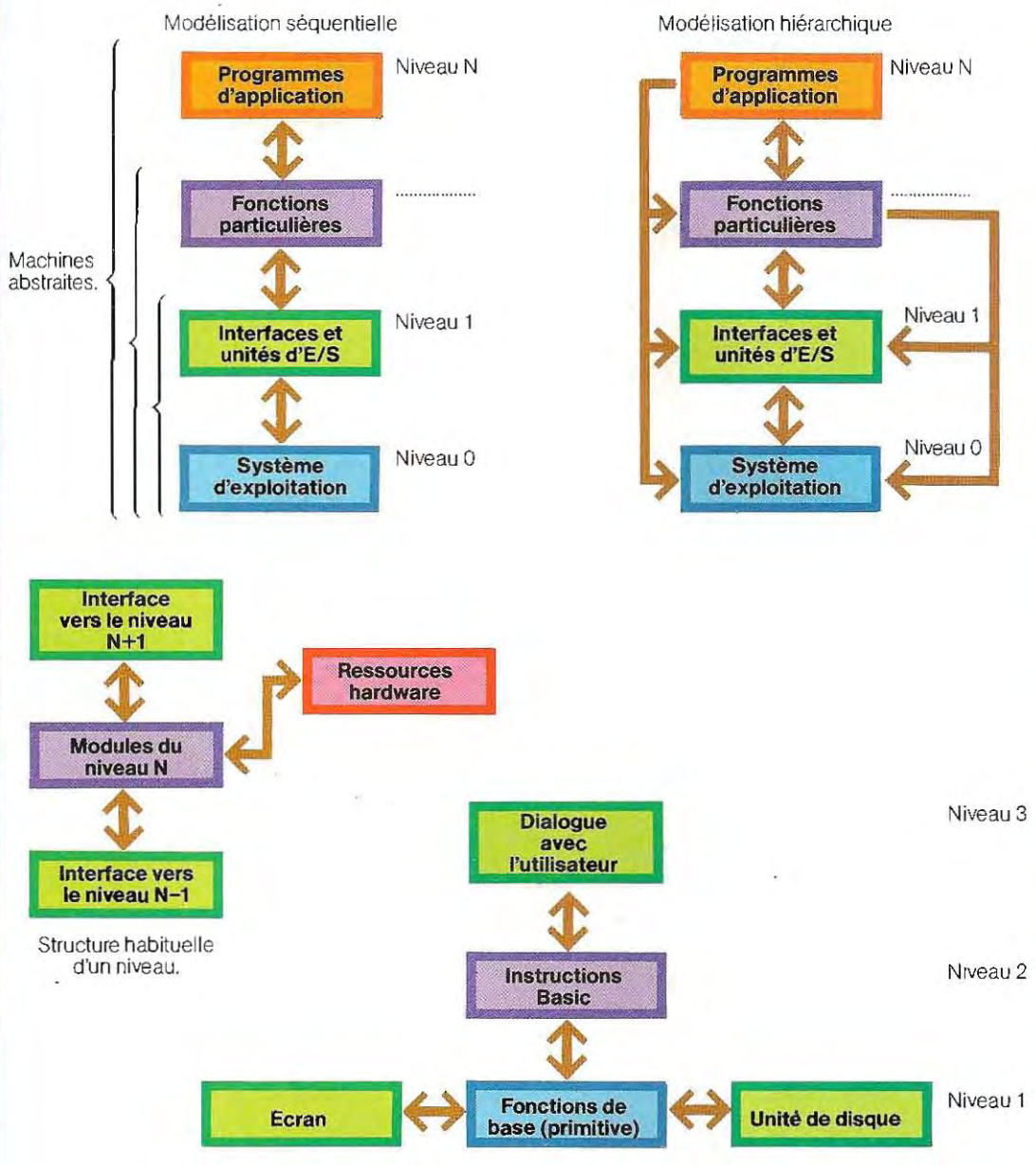
Adaptabilité. La trop grande spécificité d'un logiciel peut être avantageusement compensée par une bonne adaptabilité, c'est-à-dire par la possibilité d'ajouter ou de supprimer facilement des modules. On peut ainsi généraliser un programme ou l'adapter à de nouvelles applications sans aboutir à une structure trop complexe.
Les progiciels présentent souvent une grande souplesse, essentiellement assurée par leur segmentation en sous-programmes, voire en programmes chaînés. On peut alors ajouter ou remplacer des modules au fur et à mesure de l'évolution des besoins. Cette méthode renforce également la fiabilité, dans la mesure où elle permet d'échelonner dans le temps l'intégration des différentes fonctions.

Facilité de maintenance. La correction ou la mise à jour d'un logiciel doit être relativement simple. Cette qualité se rattache également à la fiabilité, car un programme facile à corriger est toujours plus fiable.

L'architecture du logiciel

Un logiciel d'application réunit un ensemble d'opérations distinctes mais corrélées, dont chacune peut être traitée dans un programme, ou dans une ou plusieurs routines. L'**architecture logicielle** du système vise à rassembler de façon cohérente tous ces modules en une procédure complète.
Cette approche nécessite, comme nous l'avons vu, la décomposition du programme principal en un certain nombre de sous-programmes, chacun devant à son tour se segmenter en niveaux inférieurs.
Cette modélisation en différents niveaux peut s'envisager de deux manières. On peut dire que la première méthode est séquentielle, car chaque niveau ne communique qu'avec l'étage précédent et l'étage suivant, tandis qu'avec la seconde méthode, la communication reste possible, non seulement entre niveaux contigüs, mais également entre un étage et tous ceux qui lui sont inférieurs. Sur les schémas qui représentent ces deux structures (page 771), le contenu mentionné pour chaque niveau n'a qu'une valeur d'exemple.
Cette analyse très générale s'avère surtout utile pour la réalisation de logiciels extrêmement complexes, destinés à tourner sur de gros systèmes, mais on peut également l'em-

ARCHITECTURE DU LOGICIEL ET NIVEAUX D'ABSTRACTION



Représentation schématique
d'un programme Basic sur trois niveaux.

ployer avec profit sur un micro-ordinateur.
Chaque niveau d'abstraction permet de concevoir l'ensemble **(hardware + software)** comme une **machine abstraite,** dont les possibilités sont fonction de ce niveau: au degré 0, la machine abstraite ne comporte (outre le matériel) que le système d'exploitation. Celle du niveau 1 possède en sus les interfaces entre l'extérieur et les programmes système. La complexité continue d'augmenter à chaque degré, le niveau le plus haut étant celui du dialogue avec l'utilisateur.
Ainsi, par exemple, dans un programme de lecture d'un fichier, le niveau d'abstraction le

plus élevé correspondra au module qui saisit au clavier le numéro de l'enregistrement que souhaite lire l'opérateur. L'instruction GET (lecture de l'enregistrement) représente le niveau immédiatement inférieur. Il est lié à un niveau encore plus bas (traduction en Assembleur), lui-même lié au niveau 0 (fonction primitive d'accès au disque).
Exposons maintenant les principales propriétés de ce type d'architecture, dont certaines transparaissent dans notre exemple.

- Les caractéristiques d'un niveau ne sont pas connues aux niveaux inférieurs ;
- les échanges entre niveaux s'effectuent par le biais d'interfaces prédéfinies ;
- chaque machine abstraite possède des ressources propres, dont elle doit assurer le fonctionnement par interfaçage avec les ressources des autres niveaux ;
- les échanges entre niveaux ne peuvent porter que sur certaines données (les arguments des fonctions Basic, par exemple), tandis que les autres données ne peuvent circuler qu'entre les différents éléments d'un même niveau.

Ce type d'architecture, fondé sur la hiérarchisation des opérations à exécuter en différents niveaux de complexité, doit se compléter par une structure combinant trois grands types de communication :

- transfert du contrôle sans transfert de données (GOTO, par exemple) ;
- transfert du contrôle accompagné d'un transfert de données (sous-programmes) ;
- transfert de données exclusivement (par les ports d'accès).

On peut considérer le transfert de données entre deux modules (dont chacun dispose de plusieurs ports d'accès) comme un processus asynchrone. Le module qui doit envoyer des données à un autre les transmet au port d'entrée de ce dernier, qui ne les lit que lorsqu'il est prêt.
C'est précisément parce que chaque module opère indépendamment de l'autre qu'on dit que le processus (SEND/RECEIVE) est asynchrone. Dans ce type d'architecture, les éléments les plus importants sont les modules d'interfaçage, et plus particulièrement ceux qui gèrent le dialogue avec l'utilisateur.

C'est leur manque de souplesse qui explique une grande part des problèmes de fonctionnement des logiciels. Il faut adapter leur structure à chaque cas particulier, en observant quelques règles essentielles :

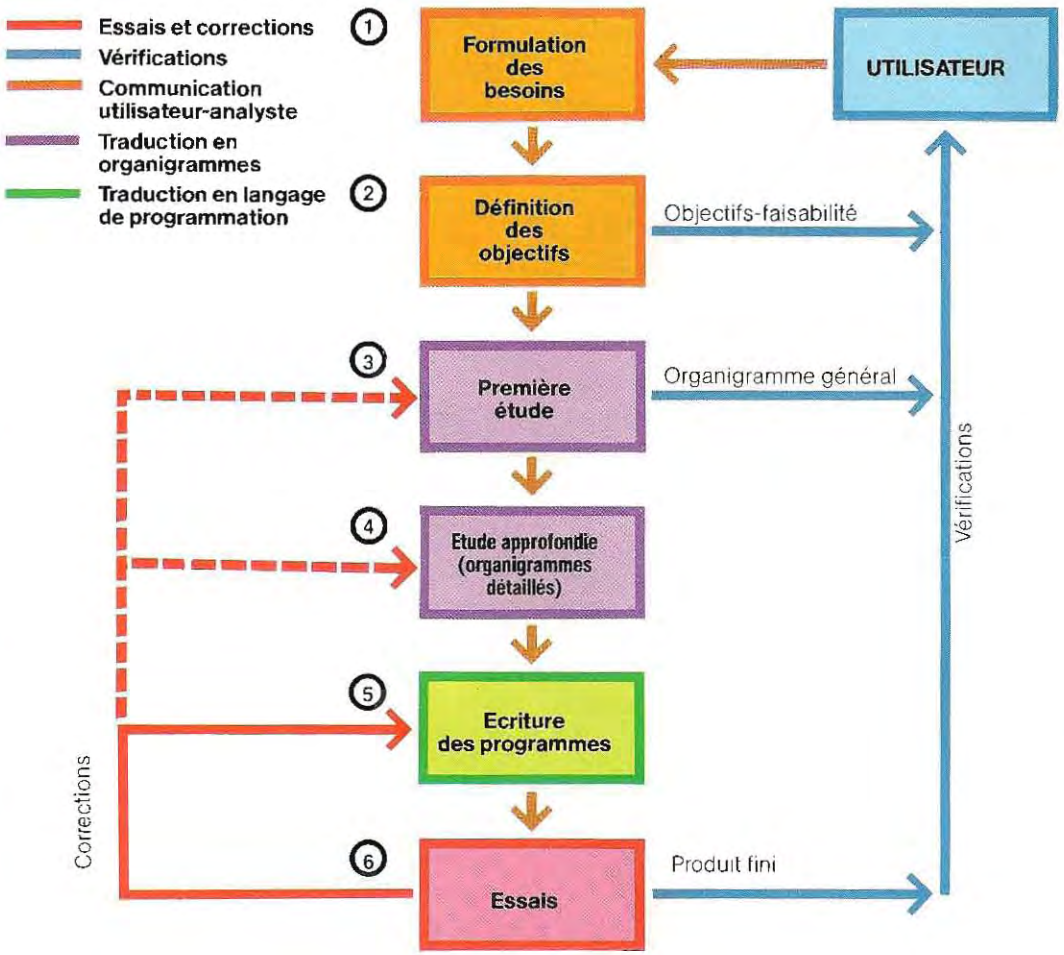
1 / les messages doivent être clairs, à la fois concis et complets. Si la forme abrégée peut convenir pour un message qui n'apparaît que rarement, un message qui revient souvent devra, surtout s'il s'adresse à des personnes menant plusieurs activités de front, être bref mais très explicite, de façon à pouvoir se comprendre instantanément, sans effort de réflexion ;
2 / il faut réduire au maximum les saisies de données, de commandes et de paramètres par l'utilisateur, en recourant le plus possible aux valeurs par défaut. On gagne ainsi un temps précieux lors de l'exécution ;
3 / les messages en entrée et en sortie doivent toujours être visualisés avec la même présentation, afin de respecter une plus grande continuité dans le dialogue avec l'utilisateur ;
4 / dans une procédure très complexe, il est bon de prévoir une commande qui active l'affichage d'un mode d'emploi du programme. En effet, il arrive souvent, notamment lors de l'apprentissage du logiciel, que l'opérateur ne sache plus très bien comment procéder. Des explications succintes fournies en temps réel lui éviteront de provoquer des erreurs en tapant des commandes non prévues. Ce module d'aide est couramment désigné par le terme anglais help.

La modélisation

On peut envisager le processus de réalisation d'un logiciel d'application comme une concertation à plusieurs niveaux. L'utilisateur explique ses besoins à l'analyste, qui décompose le problème à résoudre en opérations élémentaires et les transmet à son tour, sous forme d'organigrammes, au programmeur. Celui-ci les traduit en langage informatique, constituant le logiciel proprement dit. Au cours de sa réalisation, le projet passe donc par deux phases distinctes, comportant chacune plusieurs étapes de traduction :

- la première phase aboutit à un modèle macroscopique, définissant les objectifs et les grandes lignes du projet ;

REALISATION D'UN LOGICIEL : MODELE MACROSCOPIQUE



— la seconde phase (modèle microscopique) décompose chaque fonction (dont la nécessité est apparue pendant la première phase) en opérations élémentaires.

Les deux modèles peuvent correspondre chacun à la succession de ces deux phases, appliquées à un degré inférieur.
On obtient ainsi une modélisation sur deux niveaux – macroscopique et microscopique – qui serviront de guide, lors de la préparation, mais surtout pendant le travail de recherche des erreurs.
La plupart des erreurs sont commises au cours de l'une des opérations de traduction prévues dans les modèles. Il suffit de reprendre ceux-ci pour en trouver l'origine.

Le modèle macroscopique. Le schéma ci-dessus représente sous une forme simplifiée le modèle macroscopique du processus de développement d'un logiciel.
Lors de la première étape, l'utilisateur explique ses besoins à l'analyste (1). La découverte tardive d'une erreur commise au cours de cette phase peut coûter très cher, car sa correction nécessitera une modification importante de la structure même du logiciel. Il faut ainsi s'entourer dès le début du maximum de précautions, sans oublier que l'utilisateur lui-même parvient rarement à exprimer avec précision ses propres exigences. C'est donc à l'analyste de préparer soigneusement l'entretien et de le conduire judicieusement afin que rien ne puisse être omis ou mal compris.