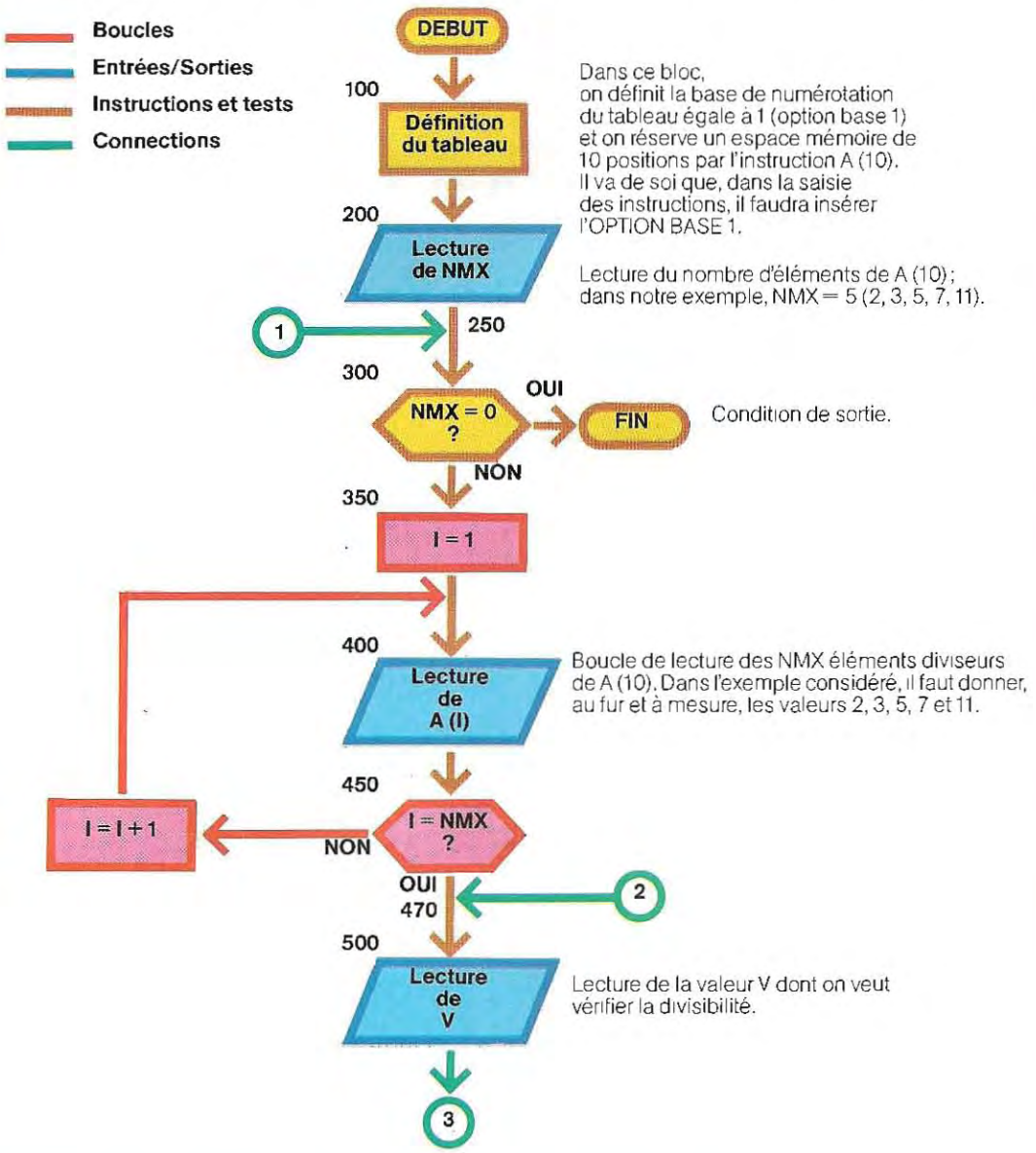
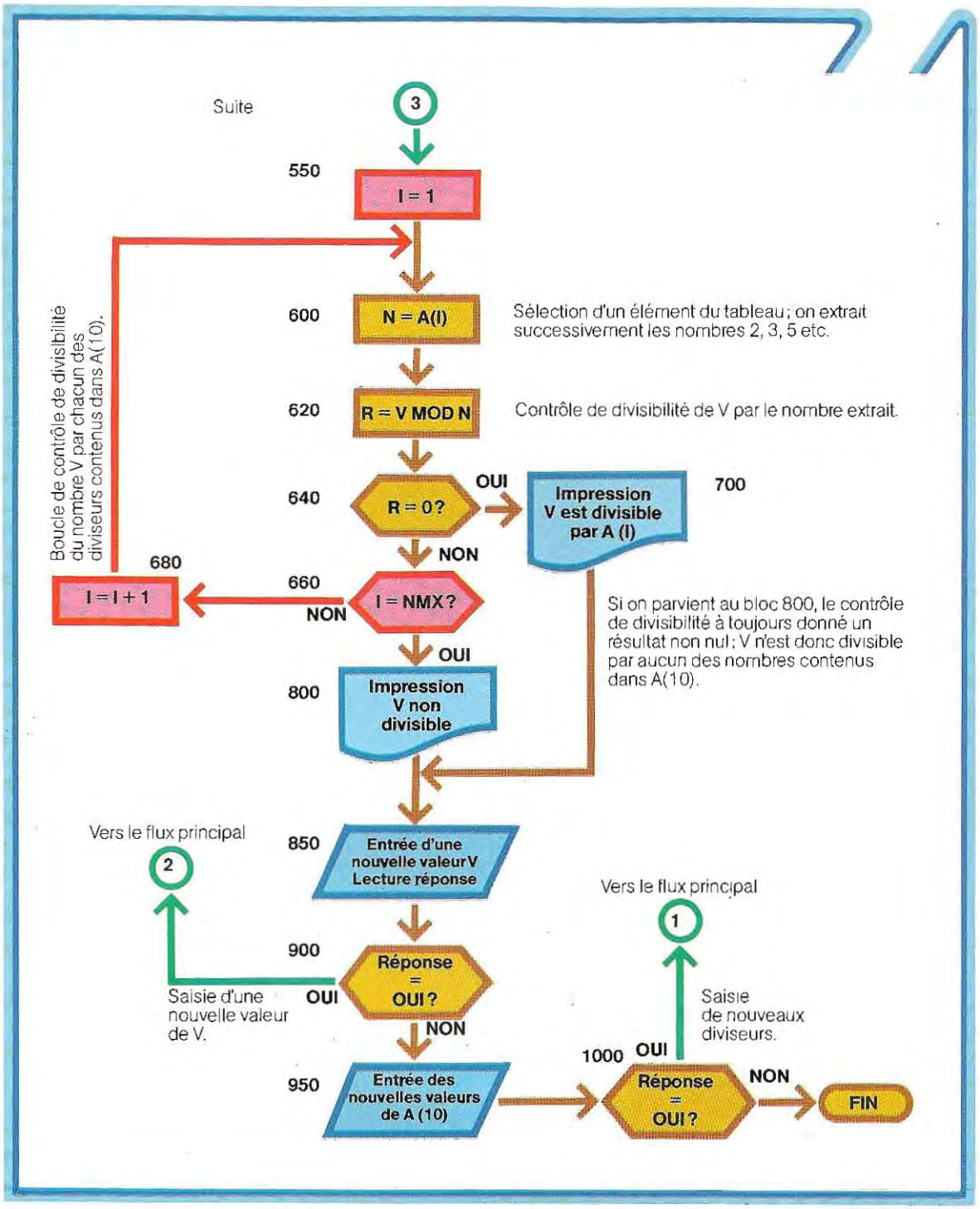
5 / Sur l'organigramme précédent, le même calcul et le même test sur zéro sont répétés cinq fois. Pour les écrire une seule fois, nous allons paramétrer les fonctions utilisées (MOD et test zéro). De cette façon, dans des situations différentes, il suffira d'actualiser les valeurs numériques (paramètres) propres au calcul en cours.
Sur ce point, le second organigramme (pages 399 et 400) peut être considéré comme optimisé de façon satisfaisante.

PROCEDURE DE CONTROLE DE DIVISIBILITE (SECONDE VERSION)





programme, afin d'éviter des instructions trop complexes et peu lisibles.
Une application possible de cette instruction ON ... GOTO ... est suggérée dans le sous-programme de calcul des surfaces de certaines figures planes (voir page 373).

Sur le listing correspondant, on s'est borné à indiquer que la sélection du type de figure se fait au moyen du paramètre K. Pour K = 1, on veut le carré (instruction 1660) ; pour K = 2, le rectangle (instruction 1680), et ainsi de suite jusqu'à la valeur K = 5, correspondant au

cercle (instruction 1770).
La sélection se programme par l'instruction :

ON K GOTO 1660, 1680, 1710, 1740, 1770

Le schéma ci-dessous présente l'organigramme complet et le listing comprenant la nouvelle instruction. Tous les commentaires précédents ont été supprimés, et les instructions de calcul (1670, 1700, etc.), qui restent inchangées, n'ont pas été reproduites. La ligne 1656 est un piège (trap) qui arrête le programme si une valeur de K erronée (hors de l'intervalle 1-5) survient.

Instructions conditionnées

On dit d'une instruction qu'elle est conditionnée quand son exécution dépend de la valeur de certaines variables.
Toutes les instructions peuvent être conditionnées sous la forme suivante :

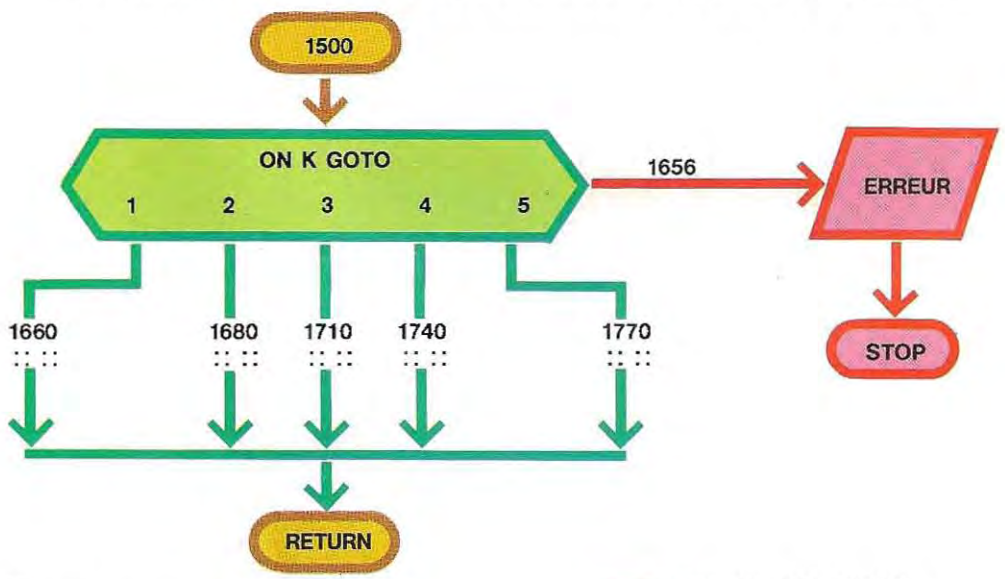
IF condition THEN instruction

Le mot IF (si) et THEN (alors) indiquent à la machine que l'instruction est subordonnée à une condition ; il faudra donc évaluer et vérifier cette condition à l'exécution. L'instruction qui suit le code THEN sera alors exécutée ou sautée.
La condition peut être représentée par une simple variable, par un calcul arithmétique (dans ces deux cas, la condition est vraie si sa valeur est différente de zéro), ou par une expression logique. D'autres exemples typiques seront donnés page 406.

IF V < > 0 THEN PRINT "Non zéro"
Si V est différent de zéro, imprimer la phrase "Non zéro"
IF A > B THEN C = A + B
Le calcul C = A + B est effectué seulement si A est supérieur à B
IF A = B THEN 1020
Si A = B, le programme saute à l'instruction 1020.

Cette dernière instruction est un exemple de saut conditionnel (voir instruction GOTO) et peut être écrite sous une autre forme, plus explicite : IF A = B GOTO 1020.
On remarquera que seule l'instruction GOTO

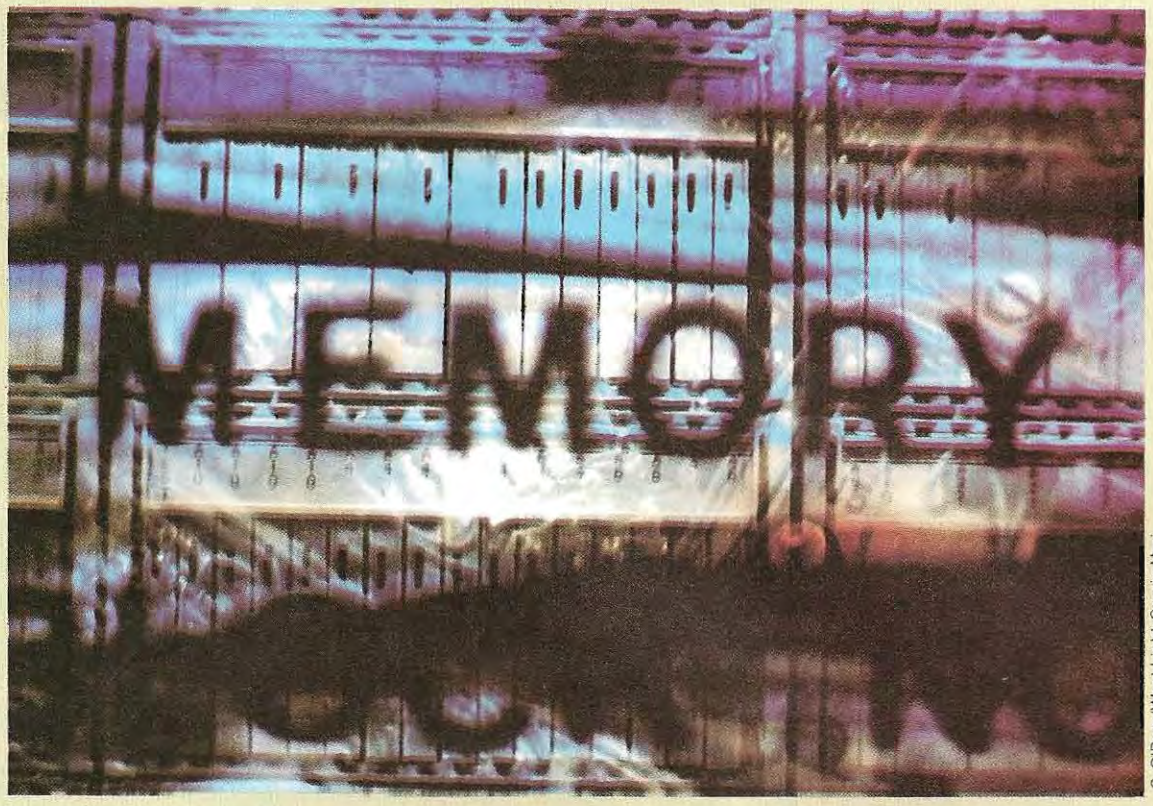
ORGANIGRAMME ET LISTING D'UNE INSTRUCTION ON ... GOTO ...




```
1500 '
1655 ON K GOTO 1660, 1680, 1710, 1740, 1770
1656 PRINT "ERREUR": STOP
1660 ' CARRÉ (K = 1)
.. .....
.. .....
```

Quel avenir pour l'ordinateur ?

L'enthousiasme suscité par le développement des applications de l'informatique au cours de ces dernières années a amené certains spécialistes à imaginer une évolution quasi illimitée de leurs ordinateurs.
Les résultats obtenus aujourd'hui, par rapport aux perspectives et aux ambitions qui orientèrent la recherche il y a quelques années, ont convaincu H.R.J. Grosch qu'il était temps de mettre en garde les « inconditionnels enthousiastes » contre un triomphalisme exagéré.
Le Dr Grosch a été président de l'ACM (Association for Computing Machinery) et a occupé de nombreux postes dans l'administration fédérale et dans quelques grandes compagnies américaines. Aussi, vaut-il la peine de s'arrêter un instant sur ses considérations.
J'ai passé une bonne partie de ma vie à recommander l'usage de l'ordinateur et cela en a valu la peine. Mais malgré mon enthousiasme, je suis resté prudent.
Il y a quelques années, on nous avait promis un langage de programmation indépendant de la machine et commun à tous les gros ordinateurs. On nous avait promis également la traduction du langage machine et l'apparition de systèmes de gestion intégrés. On nous avait dit que les programmeurs disparaîtraient, que les ordinateurs pourraient lire les caractères imprimés et même l'écriture, et qu'ils joueraient aux échecs mieux que l'homme.
On nous avait dit aussi qu'un gros ordinateur aurait les dimensions d'une montre et qu'il comprendrait le langage parlé. Aujourd'hui, on nous dit que le clavier tombera bientôt en désuétude, on répète que les programmeurs ne vont pas tarder à disparaître, que les ordinateurs traiteront des concepts plutôt que des données, qu'un million de processeurs reliés en série résoudront des problèmes d'une complexité inimaginable, et que de nouveaux langages spécialement étudiés permettront à nos enfants d'apprendre plus facilement et plus rapidement que par les méthodes d'enseignement traditionnelles.
Combien de ces promesses ont-elles été tenues ? Et, en ce qui concerne celles que l'on nous fait encore, combien nous faudra-t-il attendre avant que le nouvel hardware si puissant, le software et ses capacités de traitement atteignent nos bureaux, nos usines et nos écoles ?
Le premier fait à prendre en considération est que la réussite des années 1950, 1960, 1970 est arrivée un peu plus lentement qu'on ne l'attendait.
Il est vrai que pour traiter les données commerciales, il y a le Cobol, complètement standardisé et indépendant de l'unité centrale de traitement, mais aujourd'hui, vingt ans après leur apparition, même les programmes Cobol les plus simples demandent encore une certaine mise au point quand on veut les transférer d'IBM à Burroughs ou à ICL, par exemple.
Ce sont les mécanismes d'entrée/sortie et la complexité du logiciel sous-jacent au système de mise en forme qui l'exigent. Les mini-Cobol et les micro-Cobol qui existent ne sont pas du tout standardisés.
Les tentatives de traduction et de contrôle des langages en langue anglaise sont restées sans succès. Il y a trente ans c'était encore un rêve. Il y a vingt ans de nombreuses équipes y travaillaient avec ardeur dans les universités et les industries : j'en ai moi-même dirigé une. Mais le problème s'avéra trop ardu ; les résultats se faisaient attendre.
Même la réalisation d'un « dictionnaire électronique », qui serait d'une telle utilité aujourd'hui pour la vérification de l'orthographe dans les systèmes de traitement de texte (Word Processing), s'est alors révélée trop lente et trop coûteuse. Cependant, même ceux qui étaient alors conscients de ces difficultés ne voulaient pas céder. Le Pentagone finit par ne plus les subventionner et l'initiative en resta là.
Les SIG (Systèmes Intégrés de Gestion) étaient il y a dix ou quinze ans un concept à la mode.
Les chefs d'entreprise n'auraient plus eu qu'à s'asseoir face aux mises en images des données prévisionnelles et des différentes alternatives simulées qui auraient défilé sous leurs yeux à toute allure. Ce projet se révéla utopique avant même qu'on n'en parlât dans les articles de fond du Business Week. L'enthousiasme suscité apporta toutefois crédits et prestige au projet de réalisation des banques de données. Aujourd'hui, c'est elles qui fournissent toutes les informations que nous désirons recevoir.



C. O'Rear/West Light-Grazia Neri

Les possibilités d'emploi de l'ordinateur sont potentiellement illimitées, mais cela dépend bien évidemment des capacités de leurs mémoires elles-mêmes.

Cependant, les relations entre l'ensemble des données financières et les réalisations variaient selon la situation et le moment ; les décisions relatives au choix de l'emplacement d'une usine dépendaient en partie des problèmes familiaux du président, et la recherche de marchés extérieurs dépendait aussi d'événements politiques ou religieux.

Il est vrai que les ordinateurs jouent très bien aux échecs et qu'ils ont largement dépassé le jeu de dames, mais ils jouent comme en ont décidé leurs programmeurs, même s'ils sont plus patients, plus exhaustifs et extrêmement rapides. Ils n'ont pas inventé une nouvelle façon de jouer aux échecs : leurs coups sont encore ceux étudiés par les hommes.

Bien sûr, l'unité centrale du plus grand ordinateur des années 1950 peut tenir sur une puce de la taille d'une écaille de poisson, mais les disques et les dérouleurs de bandes magnétiques occupent toujours une bonne partie des salles des machines.

Parfois, quand le succès est à portée de la main, il ne sert plus à rien. Par exemple, la lecture optique des caractères, qui dépend presque exclusivement du perfectionnement du hardware, est possible. L'ordinateur peut déjà pratiquement lire tout document imprimé ou même bien dactylographié.

Mais tant de données ont été désormais accumulées par les machines que la capacité de les enregistrer sous une forme nouvelle n'a plus guère d'importance.

Dans ma liste de promesses pour le futur, j'ai mentionné que les plus enthousiastes avaient prédit la disparition prochaine des programmeurs ; c'est un euphémisme pour dire que les espoirs des premiers experts en génie logiciel ont été déçus. Il existe aujourd'hui des milliers de programmeurs et leur nombre ne cessera d'augmenter.

Tout ordinateur appliqué à la gestion, qu'il s'agisse d'un mini ou d'un micro, est potentiellement sous la dépendance d'un programmeur qui commet l'imprudence de s'aventurer dans les méandres du logiciel. Il y a un abîme entre la façon de travailler d'un ordinateur et celle des êtres humains. Cet abîme est comblé des milliers de fois chaque jour par des programmeurs capables, ou médiocres,

A. Nogues/Sygma-Grazia Neri

Grâce à l'ordinateur, tous les travaux de bureau sont simplifiés.

et par le personnel chargé du fonctionnement, de la manipulation et de la saisie des données.

Que peut apporter l'ordinateur aujourd'hui ? Et, plus précisément, que peut-il offrir de neuf et de précieux en dehors des domaines de la comptabilité et des techniques de l'ingénieur, où l'on peut déjà lui faire confiance ? La première réaction est de se demander : qu'y a-t-il qui ne fonctionne pas bien dans la comptabilité et dans les techniques ? Dans le monde entier, à New York, comme à Caracas, à Paris, comme dans le Punjab, des applications pratiques et rémunératrices attendent d'être réalisées dans des secteurs que nous connaissons déjà très bien. Elles nécessitent un personnel qui s'occupe de l'analyse des programmes, des experts en logiciel, des équipes chargées du fonctionnement et de la manipulation. Pour améliorer, dans le monde entier, la sélection, la formation et la gestion de ces organismes, il nous faut un hardware plus rapide, plus performant, plus fiable et moins coûteux. Nous l'aurons plus tôt que nous ne l'espérons et avant même d'être en mesure de l'assimiler. Nous pourrions supprimer ce problème, ne serait-ce que pour des raisons de choix.

En effet, les inventeurs, les producteurs, les vendeurs, les experts vous diront « achetez le mien » et il y a des centaines de possibilités et des milliers de combinaisons. En tout cas, il y a une particularité que les vétérans n'ignorent pas et que les victimes d'aujourd'hui devraient connaître : presque tout fonctionne. Vous pourrez, parfois, en être pour vos frais, ou il peut arriver que vous ayez choisi un cabinet de consultants en faillite, mais, en général, l'ordinateur fonctionne vraiment. Son coût représente en outre une faible part – non négligeable, mais dans tous les cas, faible – du montant du devis qu'il faudra établir pour résoudre un problème spécifique. La phase de choix d'un problème bien posé et le fait de pouvoir compter sur un personnel compétent sont des moments autrement plus difficiles et importants que ceux du match Honeywell - IBM.

Aujourd'hui, nous pouvons nous servir d'un instrument d'une importance capitale et dont nous étions démunis il y a seulement dix ans : le microprocesseur.

Qu'il s'agisse d'une machine indépendante, d'un terminal intelligent ou d'un contrôleur de communications, le processeur « mono-chip » peut faire des prodiges.

Quand on l'utilise pour une tâche simple, c'est-à-dire si l'on ne fait pas appel à plus de un ou deux programmes, alors il excelle. Mais il arrive de plus en plus souvent que l'apparition d'un système coïncide avec celle d'autres systèmes, et c'est alors que les problèmes de compatibilité et de standardisation deviennent énormes. A ce niveau, mon point de vue diffère de l'avis général : servez-vous, si vous le pouvez, d'un grand système central, géré de façon hautement professionnelle et qui définit des coûts bien précis. Si vous ne le pouvez pas, conservez le type de gestion, les standards et les façons de programmer propres au centre, même si vous décentralisez le hardware.

Ne laissez chaque fonction et chaque disposition suivre son propre chemin qu'en dernier recours.

Les professeurs et les jeunes entrepreneurs affirment que d'ici à quelques mois – les plus honnêtes disent quelques années – tout cela

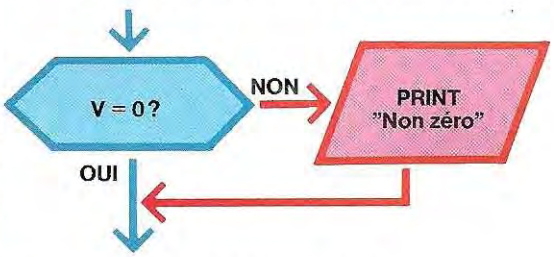
sera balayé par les machines logiques, par la compréhension de la parole et par la programmation en langage naturel. On verra apparaître des architectures nouvelles et fantastiques, des systèmes « amicaux » qui fonctionneront sans commandes ni instructions.
La raison pour laquelle j'ai tellement insisté sur le passé est que, même si nous étions alors peu nombreux, avec une expérience de base limitée, nous avions déjà entendu ces histoires, il y a dix ans. Il est vrai que certaines sont devenues réalité : nous avons le gallium, nous avons les jonctions Josephson, mais tout cela n'a que peu d'importance.
Certains objectifs, tels des interfaces complètes en langage parlé, sont, à la lettre, impossibles à atteindre.
D'autres, tels les réseaux généralisés complètement transparents, les processeurs matriciels pour le graphisme à balayage, même s'ils paraissent très ambitieux, sont abordables, mais il faudra des années avant de songer à une possible application.
D'autres objectifs encore n'auront plus tellement d'importance : le travail avec des langages comme le LOGO ou le PROLOG est fascinant, mais pour la multiplication, rien d'autre qu'une banale instruction n'est prévu. D'autre part, peu importe le caractère « amical » d'un système : il sera toujours plus facile de le connaître avec un bon manuel d'instructions.
La clé pour comprendre les possibilités offertes par l'ordinateur était généralement l'ordinateur lui-même, peu diffusé, faillible, et coûteux. Les gens le regardaient avec admiration et se battaient pour l'approcher.
Aujourd'hui, la clé des possibilités de l'ordinateur, c'est l'homme. Partout, il y a des machines et même si les experts sont très nombreux, on a surtout besoin d'analystes et de programmeurs. Le pays le plus petit et le moins avancé parmi les pays en voie de développement a besoin de centaines de ces personnes ; les pays plus développés en demandent des milliers ou des dizaines de milliers. Les pays hautement industrialisés comme le Japon ou les Etats-Unis pourront en avoir besoin par millions d'ici à la fin de cette décennie. Les universités et les collèges ne pourront en aucune façon en préparer un tel nombre. En ce moment, il ne semble pas que les écoles secondaires puissent y parvenir ni même que les universités puissent former suffisamment de professeurs pour que les écoles puissent y arriver. Cela est encore plus évident dans des pays comme Singapour où la volonté, l'argent et les facultés d'adaptation sociale sont insuffisants devant le travail qui reste à accomplir.
La solution applicable immédiatement est d'utiliser l'ordinateur pour préparer et perfectionner les analystes, les programmeurs, les ingénieurs et les opérateurs chargés de la manipulation. Nous avons à notre disposition le matériel, les logiciels et les systèmes didactiques dont la traduction dans d'autres langues que l'anglais est à l'étude. Moi-même, j'ai fréquenté des cours d'introduction au système Plato et j'ai pu noter comment, à l'aide d'un cours avancé, un expert pouvait se rendre maître rapidement d'un système difficile à acquérir. Au niveau scolaire, j'ai vu à l'œuvre ce système Plato, centralisé dans une grande université américaine, près de Capetown.
Les possibilités du microprocesseur sont par ailleurs d'une aide précieuse. Là où les coûts, l'énergie et les exigences du milieu excluent l'utilisation d'une grosse unité centrale de traitement, on peut actuellement installer des machines de bureau dotées d'un système didactique capable d'enseigner le Basic et le Fortran, la lecture et l'orthographe, l'arithmétique et la grammaire, l'algèbre et la géométrie, la comptabilité et l'ingénierie.
Ces machines n'ont pas encore accès aux énormes banques de données dont pourrait bénéficier l'enseignement en général (et plus particulièrement universitaire), et l'on se heurte également à certains obstacles culturels. Malgré tout, les progrès réalisés sur le hardware résoudront certainement le premier aspect de ce problème et je veux espérer qu'au fur et à mesure que les techniques informatiques et les types de problèmes traités par l'ordinateur se multiplieront, au moins certaines tensions d'ordre culturel s'atténueront.
Pour conclure, on peut dire que les possibilités de l'ordinateur sont aujourd'hui énormes. Il n'est plus nécessaire d'attendre les processeurs d'informations ou les jonctions Josephson. Les coûts sont désormais accessibles et ils continueront à baisser. Mais il reste les personnes – les personnes adéquates – et c'est l'ordinateur qui nous aide à les former.

(Extrait de « Le pouvoir de l'ordinateur de nos jours » de H.R.J. Grosch, Agora 1983).

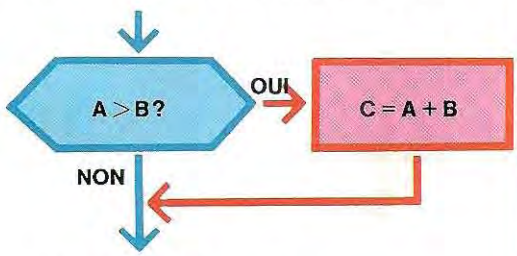
peut s'écrire sous condition (IF ... GOTO) sans le terme THEN.
Ci-dessous, sont reproduits les organigrammes des trois instructions précédentes.
Le dernier organigramme prévoit (ligne 10) la vérification de deux conditions simultanées : A = B et (AND) C > D ; dans un cas, le calcul à exécuter est F = 3 * B, dans l'autre, F = 5 * B. En ligne 10, il faut prévoir un branchement permettant de sauter l'instruction 20, faute de quoi on exécuterait, dans les deux cas, le calcul F = 5 * B. En effet, en l'absence de GOTO 126, et si les deux conditions de la ligne 10 sont vraies, on exécute F = 3 * B, mais, aussitôt après, l'instruction F = 5 * B écrase le résultat du calcul précédent. Afin d'éviter cela, il faut prévoir le saut conditionnel de la ligne 20 (si le test IF ... a donné un résultat positif), isolant ainsi le calcul F = 5 * B.

Les instructions conditionnelles sont souvent utilisées pour accélérer le déroulement des boucles. Une boucle peut être employée pour chercher une valeur déterminée dans un ensemble de données. La démarche la plus logique à adopter en ce cas consiste à programmer une boucle réalisant l'extraction successive de toutes les données présentes (en partant de la première) et leur comparaison avec la valeur désirée ; si la comparaison se vérifie, la donnée est trouvée et la boucle s'interrompt. Un indicateur initialisé à 0, prendra la valeur 1, dès que la condition de recherche sera vérifiée ; le test IF s'effectuera sur la valeur de cet indicateur, permettant ainsi de quitter la boucle.
A la page 407, on voit un exemple de recherche dans un fichier du personnel. Le fichier contient le nom et le prénom, le matri-

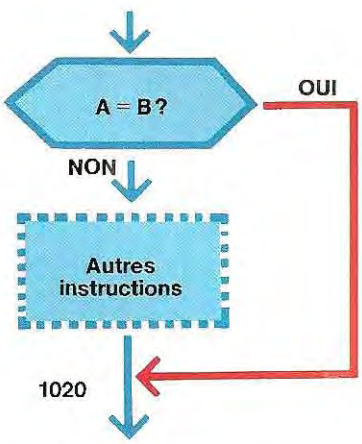
ORGANIGRAMMES DE CERTAINES INSTRUCTIONS CONDITIONNELLES



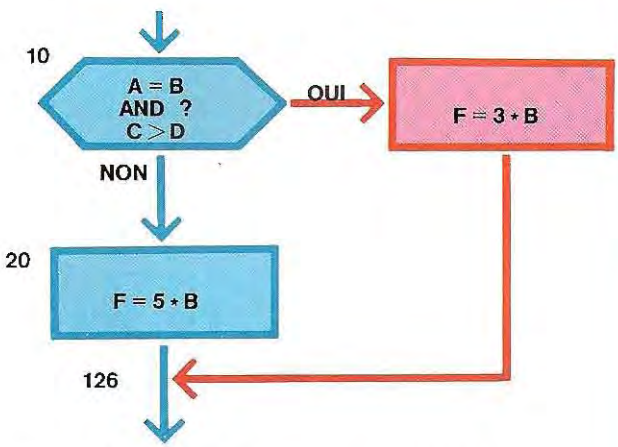
IF V < > 0 THEN PRINT "Non zéro"
(Le symbole < > signifie différent de...)



IF A > B THEN C = A + B



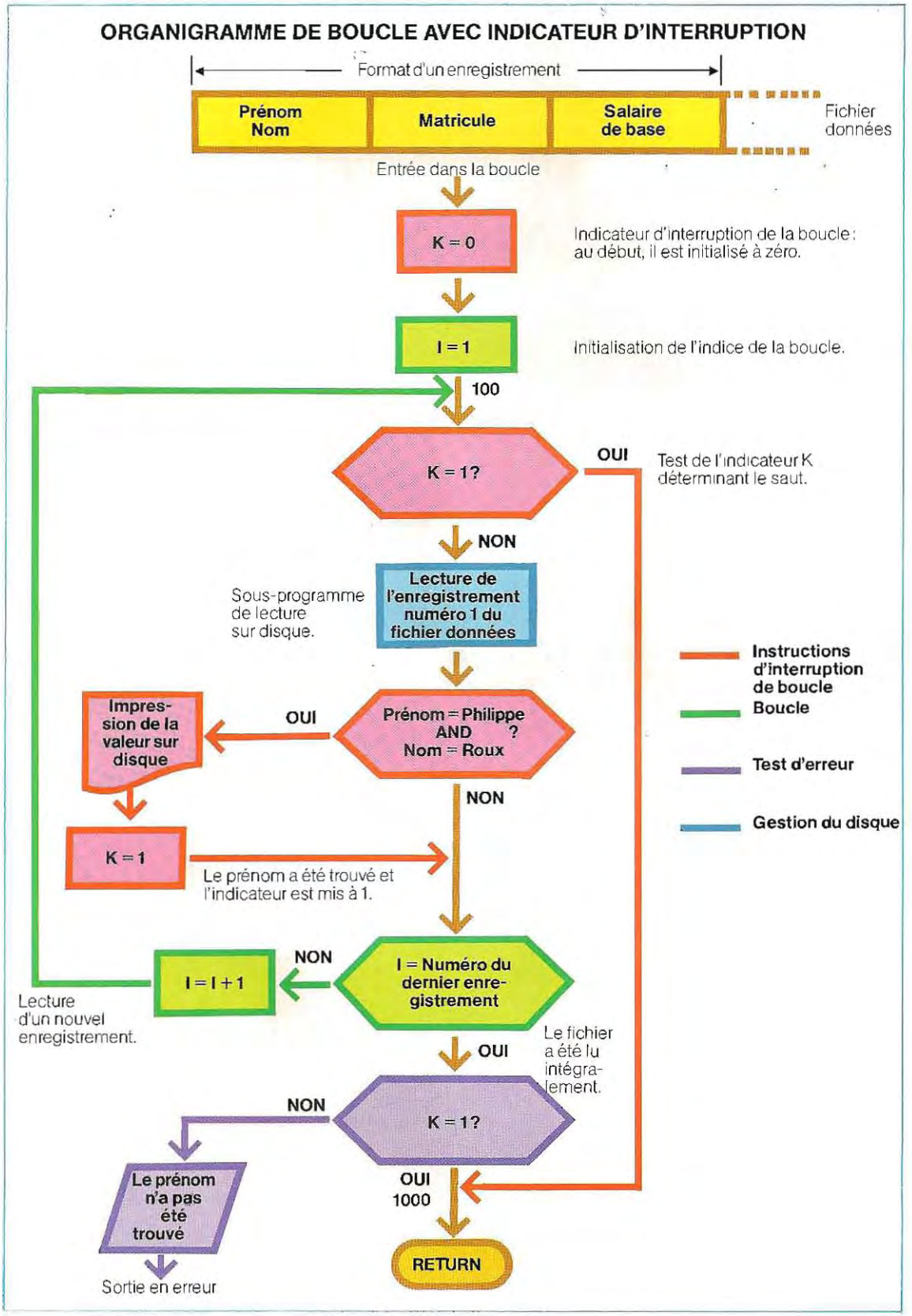
IF A = B THEN 1020
ou aussi
IF A = B GOTO 1020



10 IF A = B AND C > D THEN F = 3 * B: GOTO 126
20 F = 5 * B ' On est en 20 : la condition était fausse

126 ' On arrive de 10 : elle était vraie

ORGANIGRAMME DE BOUCLE AVEC INDICATEUR D'INTERRUPTION



cule et le salaire de base de chaque employé, et on veut connaître le salaire de Monsieur Philippe Roux. Au début, l'indicateur (K dans l'exemple) est mis à 0, cela signifiant que la recherche n'est pas terminée.
A l'entrée de la boucle (instruction 100) un contrôle testant la valeur de K est effectué : si la valeur de K est 1, on quitte la boucle ; sinon, la lecture d'un nouvel enregistrement s'effectue, ainsi que le contrôle concernant le nom lu. En cas de résultat positif, c'est-à-dire quand il y a égalité entre le nom lu et celui qu'on cherche, la donnée voulue (salaire) est imprimée et l'indicateur K prend la valeur 1.
Ainsi, lors de l'itération suivante de la boucle, l'instruction 100 s'exécute avec K = 1 et provoque un branchement vers la fin du programme (instruction 1000).
En d'autres termes, la valeur K = 1 indique que le résultat de la recherche a été positif au passage précédent et l'exécution du programme peut se terminer. Les premiers exemples du schéma de la page 406 reproduisent les formes d'instructions conditionnelles les plus simples : test sur une valeur et décision qui en découle d'effectuer ou non l'instruction. En général, une instruction conditionnelle (IF...) est double et possède deux issues possibles : la première si le contrôle est vérifié, la seconde si le test donne des résultats négatifs. Dans nos trois exemples simplifiés, la première voie se résume à une instruction, la seconde étant absente.
A l'opposé, le dernier exemple du même schéma propose les deux chemins. Il faut recourir à une instruction de saut (GOTO 126 à la fin de la ligne 10) pour ne pas exécuter l'instruction de la ligne 20, en cas de test positif sur la 10.
Cependant, cette structure n'est pas optimale. Il existe une autre instruction, qui sera préférée dans des cas analogues : il s'agit de l'instruction ELSE. Elle indique la marche à suivre dans le cas où le test de la condition n'a pas été positif.
En d'autres termes, elle restitue les deux suites possibles à cette vérification, correspondant respectivement à test = vrai (THEN), et à test = faux (ELSE).
La structure à adopter est la suivante :

IF condition à vérifier

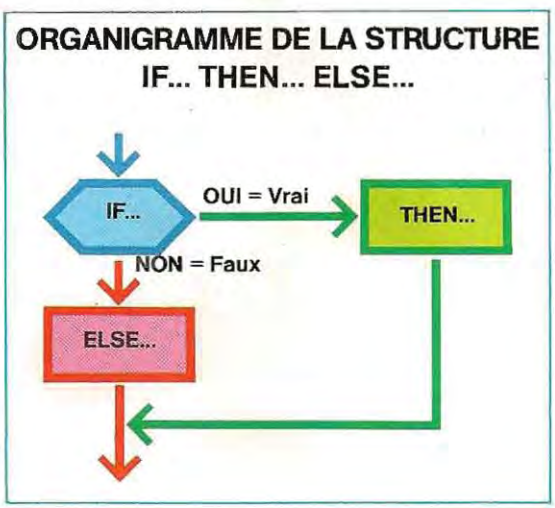
THEN instructions à exécuter si la condition est vraie

ELSE instructions à exécuter si la condition est fausse

Avec cette nouvelle instruction, la codification du dernier organigramme de la page 406 devient :

```
10 IF A = B AND C > D THEN F = 3 * B
15 'A exécuter si la condition est vérifiée.

20 ELSE F = 5 * B
25 'A exécuter dans le cas contraire.
```

Les deux instructions peuvent être symbolisées par le schéma ci-dessous.



Les instructions IF... THEN... ELSE... peuvent être alignées l'une à la suite de l'autre sur la même ligne sans être séparées par le symbole :, car elles constituent un bloc fonctionnel unique :

```
IF A > B THEN C = 1 ELSE C = 0
```

est une instruction valide, de même que

```
IF A > B THEN C = 1
     ELSE IF A = B THEN C = 0
                   ELSE C = 1
```

(qui peut aussi être écrite sur une seule ligne) analyse toutes les situations possibles pouvant se présenter dans une comparaison entre les variables A et B.

De manière plus explicite, cette dernière instruction* s'écrira comme suit :

```
IF A > B THEN C = 1
SI A supérieur à B ALORS C = 1
ELSE IF A = B THEN C = 0
SINON SI A = B ALORS C = 0
      ELSE C = -1
      SINON C = -1
```

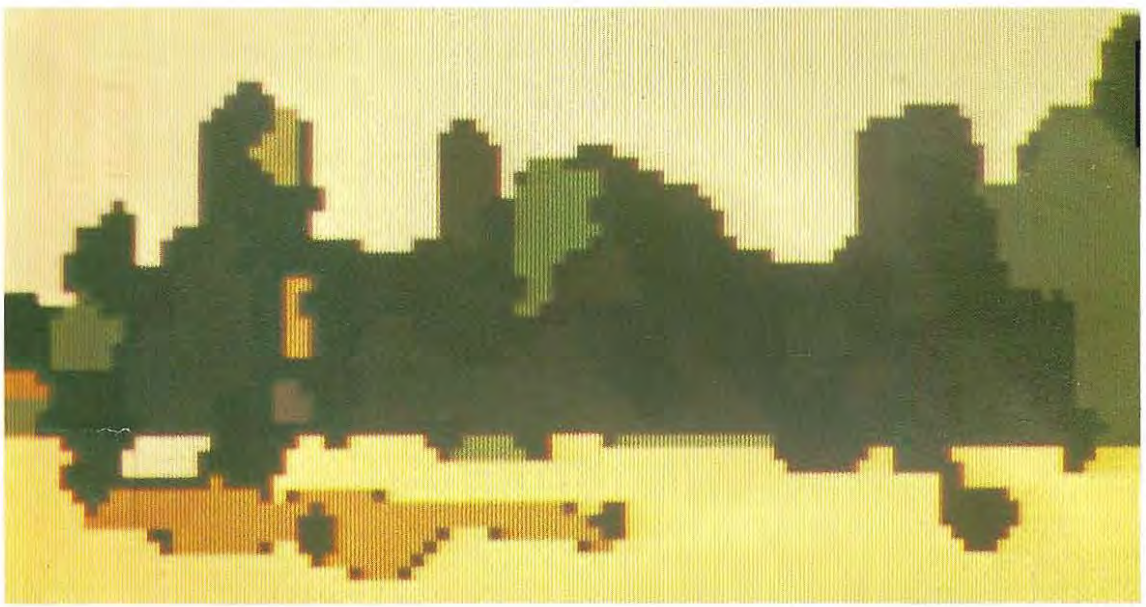
* Comme on le voit, les mots clés en langue anglaise peuvent être traduits de la manière suivante :
IF = SI THEN = ALORS ELSE = SINON

Utilisation de variables réelles dans les instructions conditionnelles

La représentation standard d'une variable réelle (c'est-à-dire qui possède une partie décimale) est la virgule flottante (floating point), suivie d'un nombre de chiffres significatifs qui dépend de la machine utilisée. Lorsqu'on affecte à la variable le résultat d'un calcul, celui-ci peut être arrondi ; l'imprécision qui en découle, si minime soit-elle, peut néanmoins fausser l'issue du test.

Pour vérifier que le résultat d'une opération est zéro, on nommera M la variable contenant ce

Deux vues de l'île de Manhattan (New York) réalisées par ordinateur en synthèse numérique.



Pierce/Wheeler Pictures-Grazia Neri

Pierce/Wheeler Pictures-Grazia Neri

Test 12

1 / Après les instructions DEFINT I-N et DEFDBL Z, quelles affectations parmi les suivantes, sont erronées?
a) IND = 3.14 b) R = 3.5 c) Z = P + 7.5 d) CIRC = I * ZA

2 / Quelles sont les valeurs affectées aux variables A, B, C, D, E, F, G par les instructions:

```
10 READ A, B, C
20 RESTORE
30 READ D
40 RESTORE 80
50 READ E, F, G
60 DATA 5, 7, 21, 40, 60
70 DATA 3, 9, 10
80 DATA 15, 20, 75
```

3 / Le programme reproduit ci-dessous contient une erreur: laquelle?

```
10 X = A + B
20 READ C, D
30 RESTORE
40 READ E, F, G
50 DATA 3, 7
```

4 / Comment modifier le programme de la question numéro 2 pour affecter aux variables E, F et G les valeurs 7, 21 et 40 (DATA 60)?

5 / Le programme listé ci-dessous contient une erreur: quelle ligne faut-il supprimer?

```
10 DEFINT I-N
20 DEFDBL Z
30 READ A, B, C
40 X = C * (A + B)
50 RESTORE
60 READ A$
70 PRINT X
80 DATA 5, 3, 2
90 DATA "Ceci est un essai"
```

Les solutions du test se trouvent page 413.

résultat, et l'instruction s'écrira :

```
IF M = 0 THEN ...
```

Sous cette forme (exacte si M était un entier), un manque de précision sur M peut engendrer des erreurs. Si, par exemple, M vaut 0.001, on assimilera, pour des raisons pratiques, cette valeur à 0, alors que la machine répondra au test IF M = 0 par un résultat négatif.

Dans une instruction conditionnelle à paramètre réel, on indiquera toujours, pour éviter ce genre d'erreur, la précision avec laquelle on veut que le contrôle soit exécuté. Dans l'exemple précédent, en écrivant :

```
IF M < 0.001 THEN ...
```

la condition est vérifiée pour toutes les valeurs de M inférieures à 0.001, ce nombre étant la précision avec laquelle le test est réalisé.

Mais l'instruction est encore inexacte: la variable M pourrait prendre des valeurs néga-

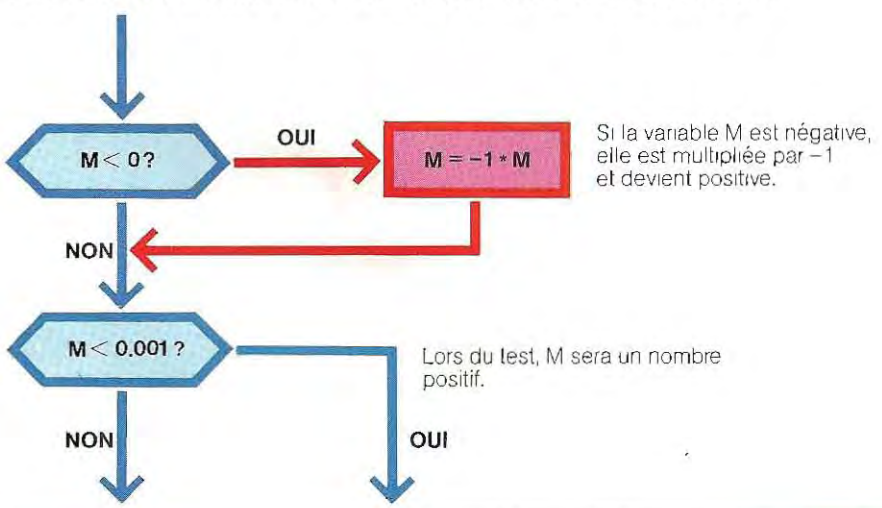
tives très éloignées de zéro, mais toujours inférieures à 0.001 (−10000 est inférieur à 0.001) et le test donnerait des résultats erronés. Pour formuler correctement cette instruction, il faut considérer la « valeur absolue » de M, c'est-à-dire sa valeur privée de signe. De cette manière (en enlevant le signe −), une éventuelle valeur négative deviendra positive et le test sera cette fois parfaitement satisfaisant.
Il existe une instruction spéciale (fonction) qui permet d'extraire la valeur absolue d'une variable : nous l'étudierons prochainement. Pour l'instant, on adoptera la méthode illustrée par le schéma au bas de cette page, et qui consiste à toujours comparer un nombre positif (M ou −M) à la valeur critique (ici 0.001).

Appel d'un sous-programme et enchaînement de programmes

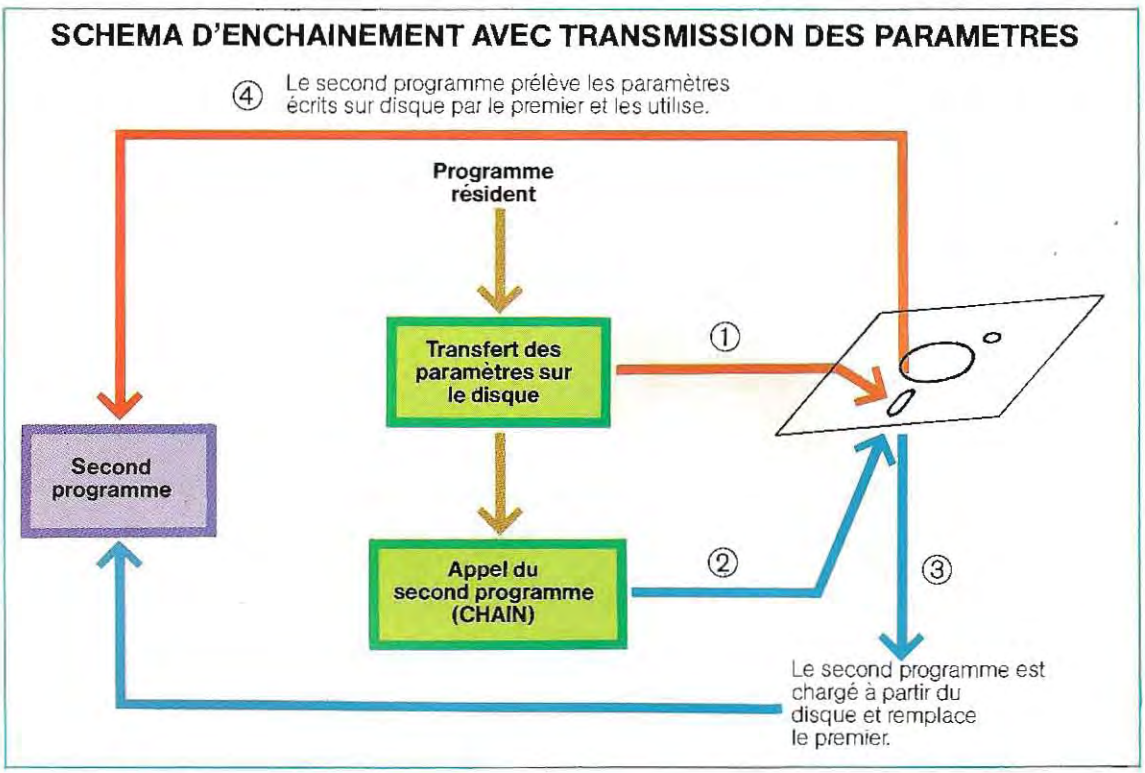
L'instruction « d'appel » d'un sous-programme est GOSUB n, n étant le numéro de la ligne à partir de laquelle débute le sous-programme (**sub**routine en anglais). Cette instruction peut être conditionnelle. Par exemple :

GOSUB 1250
Dans ce cas, le sous-programme 1250 est appelé sans aucune condition, mais, si on a :
IF A > B THEN GOSUB 1250
l'exécution saute à 1250 seulement si A est supérieur à B. Si on a :
ON M GOSUB 1250, 720, 500
l'instruction donne le contrôle à un des trois sous-programmes 1250, 720 ou 500, selon la valeur de M (voir instructions ON ... GOTO ...).
L'instruction GOSUB ... ressemble à l'instruction GOTO ..., mais ici le point d'arrivée du saut est le début d'un sous-programme, au terme duquel l'instruction RETURN ramène l'exécution au programme appelant, à la ligne suivant le GOSUB.
Les sous-programmes (appelés aussi routines) faisant logiquement partie du programme appelant, seront tous chargés en mémoire avant l'exécution. Leur utilisation, bien que recommandée pour une meilleure structuration des programmes, ne permet pas de gain en espace. Il ne faut pas confondre cette subdivision logique en routines avec le découpage physique d'un programme trop volumineux pour être intégralement contenu dans la mémoire de l'ordinateur. Dans ce dernier cas, on divisera le programme en parties plus petites, résidant sur disque, et qui seront chargées en mémoire seulement au moment de leur utilisation (voir **les systèmes d'exploitation**, page 291).
Chacun de ces modules constitue un programme à part entière ; au moment de son chargement, il remplacera en mémoire le programme précédent.
L'instruction réalisant le chargement et l'exécution d'un programme à la suite d'un autre est CHAIN « NOM », où NOM désigne le fichier dans lequel est mémorisé (sur disque) le programme à charger. Par exemple :

TEST DE LA VALEUR ABSOLUE D'UNE VARIABLE REELLE



SCHEMA D'ENCHAINEMENT AVEC TRANSMISSION DES PARAMETRES



CHAIN « ESSAI »
charge le programme ESSAI.
CHAIN « B : XY »
charge le programme XY résidant sur disque B.

L'instruction CHAIN peut aussi indiquer le numéro de la ligne à partir de laquelle doit commencer l'exécution du programme chargé. CHAIN « ESSAI », 275 charge le programme ESSAI et l'exécute à partir de l'instruction 275. Si le numéro est omis (exemples précédents), l'exécution commence à partir de la première instruction du nouveau programme.
Les programmes successivement appelés et exécutés peuvent posséder certaines variables en commun. Il faut alors transmettre (« passer ») ces paramètres de l'un à l'autre. Le transfert se réalise de trois façons différentes :

- avec un fichier intermédiaire comme support des valeurs,
- par l'option ALL,
- par l'instruction COMMON.

Dans le premier cas, le programme résident écrit dans un fichier spécial du disque tous les paramètres qui doivent être transmis au programme suivant. Une fois chargé, ce dernier lit les valeurs du fichier, afin de les utiliser.
Le schéma ci-dessus décrit les opérations que nécessite cette technique. La méthode est laborieuse, mais reste parfois la seule possible, car la majeure partie des petits systèmes d'exploitation ne proposent pas d'autre solution.
La deuxième méthode consiste à spécifier l'option ALL dans l'instruction CHAIN.
Elle permet de transmettre toutes les variables d'un programme au suivant. Par exemple,

CHAIN « NOM », 152, ALL

signifie : charger le programme NOM et l'exécuter à partir de l'instruction 152, en conservant toutes les valeurs des variables. Tous les compilateurs ne prévoient pas cette option. En général, elle est possible en Basic interprété et non en compilé.
L'emploi d'une instruction COMMON est la solution la plus classique et aussi la plus souple pour transmettre des paramètres. Elle peut être utilisée aussi bien en Basic interprété qu'en compilé, mais elle n'est disponible que sur les systèmes d'exploitation relativement évolués.

Le code COMMON déclare « communes » toutes les variables énumérées; par exemple, COMMON A, X, N définit les variables A, X et N comme paramètres communs. Une zone mémoire spéciale, qui ne sera pas écrasée par le chargement d'autres programmes, est réservée aux paramètres du COMMON.
L'instruction CHAIN possède deux autres options : MERGE et DELETE.
MERGE permet de charger le nouveau programme en « overlay », (recouvrement), c'est-à-dire comme s'il s'agissait d'un morceau (segment) du programme appelant. Ainsi, toutes les déclarations des types de variables (par exemple DEFINT ..., etc.) qui, autrement, devraient être répétées dans le programme appelé, demeureront valables. Avec l'option MERGE, les segments appelés doivent être en format ASCII.
L'option DELETE permet d'effacer le segment chargé en overlay avant d'en appeler un autre. Toutefois, de telles options ne sont généralement pas prévues dans les compilateurs.

En résumé, la formulation complète d'une instruction d'enchaînement est :

CHAIN MERGE « NOM », 1350, ALL, DELETE 185 – 2730

qui prendra la signification suivante :
charger en mode overlay le programme NOM, puis déclencher l'exécution à la ligne 1350 avec transfert de toutes les variables ; effacer ensuite les instructions de 185 à 2730.
Les sous-programmes externes, n'appartenant pas au programme principal, peuvent être écrits dans les langages différents du Basic (Fortran, Cobol, Assembleur) ; en ce cas, des instructions spécifiques assureront la transmission des paramètres et l'appel des segments. Ces instructions spéciales seront étudiées plus en détail dans le chapitre traitant des techniques de programmation en langages mixtes.

Exemple d'application : gazole ou essence ?
Lors de l'achat d'une voiture, le choix du type de moteur (à gazole ou à essence), est décisif, et dépend de nombreux facteurs, tels :

Solutions du test 12

1 / Les affectations erronées sont : a) car une variable qui commence par la lettre I est définie entière (DEFINT I-N) ; d) qui contient la variable CIRC en simple précision alors que ZA est définie implicitement en double précision (DEFDBL Z).
Une troncature (un arrondi) s'effectuera dans les deux cas.

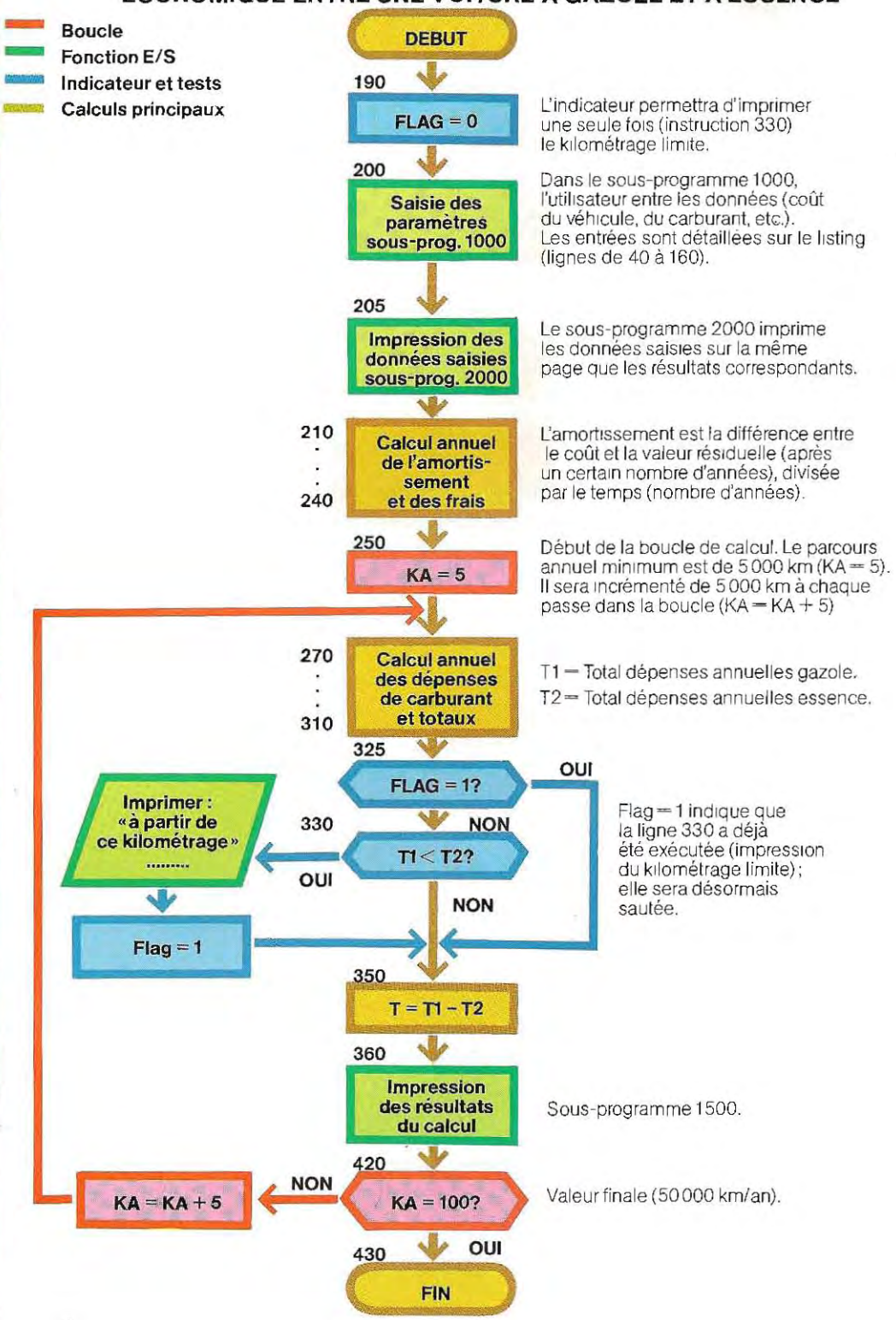
2 / L'instruction 10 exécute les affectations A = 5, B = 7, C = 21, et positionne le pointeur de DATA sur la quatrième valeur de la ligne 60. L'instruction 20 remet à zéro ce pointeur et le repositionne en début de DATA ; la valeur 5 est à nouveau affectée à la variable D (ligne 30). L'instruction 40 positionne le pointeur en début de DATA, apparaissant à la ligne 80 ; la lecture suivante (ligne 50) donnera E = 15, F = 20, G = 75.

3 / Après le RESTORE (ligne 30), le READ de la ligne 40 cherche à lire trois variables dans un DATA contenant seulement deux valeurs.

4 / Si on supprime la ligne 40, le pointeur de DATA reste sur la deuxième valeur de la ligne 60 (la première valeur a été affectée à la variable D en ligne 30). Le READ suivant (ligne 50) exécutera les affectations E = 7, F = 21, G = 40.

5 / Le RESTORE (instruction 50) ramène le pointeur en début de DATA numérique (ligne 80) ; la ligne 60 provoquera une erreur car la variable (chaîne) et les données (numériques) ne sont pas homogènes. Si l'on supprime la ligne 50, le READ de la ligne 30 positionnera le pointeur au début du DATA de chaîne (ligne 90), et la variable A$ (instruction 60) pourra être lue.

ORGANIGRAMME DU PROGRAMME DE BILAN ECONOMIQUE ENTRE UNE VOITURE A GAZOLE ET A ESSENCE



VOITURE A GAZOLE ET VOITURE A ESSENCE

```
5 ' *** COMPARAISON ENTRE VOITURES GAZOLE/ESSENCE ***
6 '
7 '   FILE - GRES
8 '
10 OPTION BASE 1   ' Les indices partent de 1
20 '
30 '
32 A$ = "£££££££££.£"
35 B$ = "£££££££££"
36 C$ = "£££££££££.££"
40 ' VALEUR DES VARIABLES
50 '  CG = Prix d'achat voiture Gazole
60 '  CE = Prix d'achat voiture Essence
70 '  KG = Kilometres parcourus avec 1 litre de Gazole
80 '  KE = Kilometres parcourus avec 1 litre d'Essence
90 '  LG = Prix du litre de gazole
100 '  LE = Prix du litre d'essence
110 '  TG = Frais divers v. gazole (ex. vignette)
120 '  TE = Frais divers v. essence (ex. vignette)
130 '  AA = Nombre d'annees d'utilisation
140 '  KA = Kilometrage annuel
150 '  VG = Valeur de revente voiture a moteur diesel
160 '  VE = Valeur de revente voiture a moteur essence
170 '
180 '
190 FLAG = 0
200 GOSUB 1000  ' SUBROUTINE ENTREE DATA
205 GOSUB 2000  ' SUBROUTINE D'EDITION
210 C1 = (CG-VG)/AA  ' Cout amortissement voiture diesel
220 C2 = (CE-VE)/AA  ' Cout amortissement voiture essence
230 S1 = C1 + TG       ' Cout amortissement voiture diesel + frais
240 S2 = C2 + TE       ' Cout amortissement voiture essence + frais
250 FOR KA = 5 TO 100 STEP 5
260 '
270 R1 = LG*KA*1000/KG   ' Depense annuelle de gazole
280 R2 = LE*KA*1000/KE   ' Depense annuelle d'essence
290 '
300 T1 = S1 + R1           ' Depense globale voiture diesel
310 T2 = S2 + R2           ' Depense globale voiture essence
320 '
325 IF FLAG = 1 GOTO 350
330 IF T1 < T2 THEN LPRINT  " A partir de ce kilometrage annuel "
332 IF T1 < T2 THEN LPRINT  " la voiture diesel est rentable " : FLAG = 1
340 '
350 T = T1 - T2
360 GOSUB 1500   ' Subroutine pour l'impression des resultats
390 '
400 PRINT " taper un caractere pour continuer "
410 S$ = INPUT$(1)
411 '
420 NEXT KA
430 END
1000 INPUT " COUT AUTO DIESEL " ; CG
1010 INPUT " COUT AUTO ESSENCE " ; CE
1020 INPUT " KM PAR LITRE DE GAZOLE " ; KG
1030 INPUT " KM PAR LITRE D'ESSENCE " ; KE
1040 INPUT " COUT LITRE GAZOLE " ; LG
1050 INPUT " COUT LITRE ESSENCE " ; LE
1060 INPUT " VIGNETTE AUTO. DIESEL " ; TG
1070 INPUT " VIGNETTE AUTO. ESSENCE " ; TE
1080 INPUT " NOMBRE D'ANNEES D'UTILISATION " ; AA
1090 '
1100 INPUT " VALEUR REVENTE AUTO DIESEL " ; VG
1110 INPUT " VALEUR REVENTE AUTO ESSENCE " ; VE
1120 '

1130 RETURN
1500 '  ***** IMPRESSION DES RESULTATS *****
1510 LPRINT " KM = " ; KA*1000
1520 LPRINT " DEPENSE ANNUELLE AUTO GAZOLE = " ; : LPRINT USING B$ ; T1
1530 LPRINT " DEPENSE ANNUELLE AUTO ESSENCE = " ; : LPRINT USING B$ ; T2
1540 LPRINT " DIFFERENCE DES DEPENSES       = " ; : LPRINT USING B$ ; T
1550 LPRINT
1560 LPRINT
1570 RETURN
2000 '
2010 '  ***** IMPRESSION DES VALEURS SAISIES *****
2020 '
2030 N$ = " ***** CALCUL ET COMPARAISON DIESEL ESSENCE ***** "
2040 D$ = "                                    TYPE DE VEHICULE "
2050 E$ = " VALEURS                          DIESEL       ESSENCE "
2060 F$ = " COUT ACQUISITION "
2070 G$ = " KM PARCOURUS AVEC 1 LITRE "
2080 H$ = " COUT DU LITRE "
2090 J$ = " FRAIS (VIGNETTE) "
2100 K$ = " NOMBRE D'ANNEES D'UTILISATION "
```

```
2120 M$ = " VALEUR DE REVENTE "
2130 LPRINT N$ : LPRINT
2140 LPRINT D$ : LPRINT
2150 LPRINT E$ : LPRINT
2160 LPRINT F$; : LPRINT TAB(44) : LPRINT USING C$ ; CG ; CE : LPRINT
2170 LPRINT G$; : LPRINT TAB(44) : LPRINT USING A$ ; KG ; KE : LPRINT
2180 LPRINT H$; : LPRINT TAB(44) : LPRINT USING C$ ; LG ; LE : LPRINT
2190 LPRINT J$; : LPRINT TAB(44) : LPRINT USING C$ ; TG ; TE : LPRINT
2200 LPRINT K$; : LPRINT TAB(44) : LPRINT USING B$ ; AA ; AA : LPRINT
2220 LPRINT M$; : LPRINT TAB(44) : LPRINT USING C$ ; UG ; UE : LPRINT : LPRINT : LPRINT
2230 RETURN
```

```
***** CALCUL ET COMPARAISON DIESEL ESSENCE *****

                                        TYPE DE VOITURE

VALEURS                                 DIESEL      ESSENCE

COUT D'ACQUISITION                     90000.00     60000.00

KM PARCOURUS AVEC 1 LITRE                 22.7         13.4

COUT DU LITRE                             3.67         4.34

FRAIS (VIGNETTE)                        330.00       225.00

NOMBRE D'ANNEES D'UTILISATION             5            5

VALEUR DE REVENTE                      30000.00     15000.00


KM = 5000
DEPENSE ANNUELLE AUTO GAZOLE   =     13182
DEPENSE ANNUELLE AUTO ESSENCE  =     11068
DIFFERENCE DES DEPENSES        =      2114


KM = 10000
DEPENSE ANNUELLE AUTO GAZOLE   =     14035
DEPENSE ANNUELLE AUTO ESSENCE  =     12912
DIFFERENCE DES DEPENSES        =      1123


KM = 15000
DEPENSE ANNUELLE AUTO GAZOLE   =     14887
DEPENSE ANNUELLE AUTO ESSENCE  =     14755
DIFFERENCE DES DEPENSES        =       132


A partir de ce kilometrage annuel, la voiture diesel est rentable
KM = 20000
DEPENSE ANNUELLE AUTO GAZOLE   =     15740
DEPENSE ANNUELLE AUTO ESSENCE  =     16598
DIFFERENCE DES DEPENSES        =      -858


KM = 25000
DEPENSE ANNUELLE AUTO GAZOLE   =     16592
DEPENSE ANNUELLE AUTO ESSENCE  =     18441
DIFFERENCE DES DEPENSES        =     -1849
```

- le prix respectif des deux types de voiture ;
- le coût de leur vignette et de leurs frais fixes respectifs ;
- les dépenses en carburant ;
- la cotation à l'argus de chacun des deux types de voiture.

Le programme de bilan économique pour les deux voitures devra tenir compte de tous les facteurs cités, en considérant aussi la marque et le modèle choisis.
L'organigramme de la page 414 fournit, en fonction de la valeur des paramètres caractérisant la dépense (prix, frais fixes, consommation), le coût total annuel pour différents parcours kilométriques, et met en évidence le kilométrage minimum à partir duquel le moteur diesel est le plus avantageux.
Le programme correspondant est listé pages 415 et 416. Un exemple de calcul l'accompagne, dans l'hypothèse de deux voitures coûtant respectivement 90 000 et 60 000 francs à l'achat, et pouvant être revendues 30 000 et 15 000 francs après cinq ans. Comme on peut le voir, l'intérêt économique se situe en faveur du véhicule diesel et pour un parcours annuel minimum de 15 000 kilomètres.
En modifiant les conditions initiales lors de la saisie des données, on obtiendra une nouvelle estimation, par exemple dans le cas d'un changement de cylindrée. La décision d'achat tiendra compte du meilleur rapport qualité-prix.

Les diagnostics d'erreurs

Lorsqu'on écrit un programme, il est inévitable, au moins dans la première phase de rédaction, de commettre des erreurs. Le travail de mise au point présente des difficultés qui augmentent avec le nombre des instructions et leur complexité. Pour cette raison, on cherchera à structurer les programmes en les subdivisant en sous-programmes, chacun pouvant alors être exécuté séparément.
Les techniques de test des programmes seront traitées par la suite. Ici, nous présenterons seulement les instructions particulières de recherche des erreurs.
On distinguera deux catégories principales :

- les erreurs relevées en phase d'écriture du programme ;
- les erreurs relevées en phase d'exécution du programme.

Les erreurs du premier type, appelées aussi erreurs de syntaxe, proviennent d'une formulation inexacte des instructions, par exemple d'une faute de frappe ou du non-respect des espaces entre deux mots. Elles sont reconnues immédiatement et signalées par l'interpréteur Basic. Leur correction est facile : il suffit de récrire l'instruction sous sa forme exacte. Il faut toutefois se rappeler que ce premier diagnostic nous assure de l'exactitude formelle des instructions, mais pas du bon fonctionnement du programme en phase d'exécution.
Par exemple, lors de la programmation d'une boucle, on peut oublier son instruction de fermeture (NEXT). A ce niveau de diagnostic, l'instruction initiale (FOR ... TO ...) est formellement exacte, et l'interpréteur ne signale aucune erreur. L'omission du NEXT ne sera mise en évidence que par la suite, lors de l'exécution du programme.
Les erreurs du second type sont plus complexes et n'apparaissent qu'à l'exécution du programme. Les méthodes qui s'appliquent à leur recherche et à leur correction dépendent principalement de ce qui les a causées.
Les principales erreurs relevées en phase d'exécution peuvent être :

- des erreurs de programmation ;
- des erreurs de logique ;
- des erreurs dans le déroulement des calculs ;
- des erreurs système ;
- des erreurs commises par l'utilisateur lors de la saisie des données.

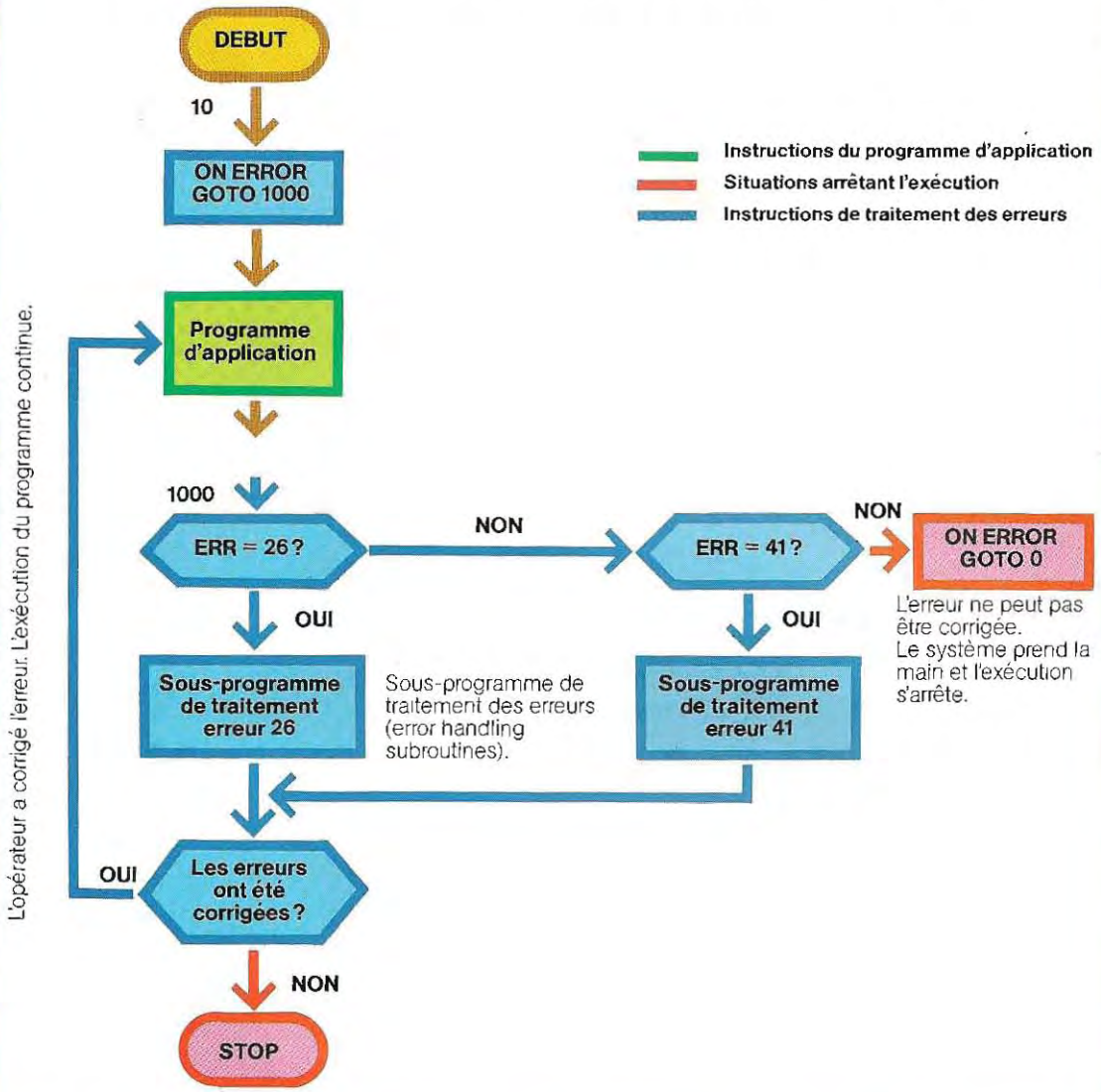
Les **erreurs de programmation** proviennent d'une mauvaise utilisation des instructions. Citons quelques exemples comme l'oubli de la fermeture d'une boucle, l'omission de l'instruction RETURN à la fin d'un sous-programme, une incohérence entre le nombre de variables d'une instruction READ et le nombre de données de l'instruction DATA associée, un numéro de ligne inexistant dans l'instruction GOTO ..., etc.
Normalement, l'interpréteur Basic possède un bon module de diagnostic, et les messages qui s'affichent à l'écran suffisent à l'identification de l'erreur.
Les **erreurs de logique** sont souvent très complexes, et ne génèrent pas de message d'erreur. Leur recherche passe obligatoirement par une analyse attentive des organigrammes, ou par l'exécution d'un cas déjà résolu manuellement. La comparaison entre les résultats obtenus par la machine et ceux calculés à la main peut aider à repérer l'endroit du programme contenant l'erreur. Le mode « trace » (TRON) constitue un autre moyen de recherche des erreurs de logique. En lançant un programme après avoir tapé la commande TRON, on verra apparaître à l'écran les numéros des instructions qui seront successivement exécutées. De cette manière, on suivra le déroulement du programme en vérifiant que l'exécution se poursuit normalement.
Les **erreurs survenant dans l'exécution des calculs** sont semblables aux précédentes. Elles proviennent d'une utilisation incorrecte des parenthèses ou des opérateurs, ou de l'utilisation d'un type de variable inapproprié. Si, par exemple, une variable déclarée comme entière (avec l'instruction DEFINT ...) prend une valeur supérieure à 32767 (ou inférieure à – 32768), (type erroné, débordement), le système donne son diagnostic. Dans les autres cas, il appartiendra au programmeur seul de rechercher la cause première des erreurs.
Les **erreurs système** sont imputables à un mauvais fonctionnement de la machine, ou à des causes accidentelles extérieures,

comme une mauvaise manipulation des mémoires de masse (l'opérateur, par exemple, oublie d'insérer une disquette, ou se sera trompé d'unité). On reviendra plus tard sur ce dernier type d'erreur.
Un programme bien conçu comportera au moins une routine de traitement des erreurs (diagnostic et, éventuellement, corrections). On accèdera à ce sous-programme de deux façons : par l'instruction d'appel ON ERROR GOTO ... (pour les erreurs détectées par le système), ou par des instructions de diagnostic spécifiques (écrites par le programmeur).
L'instruction ON ERROR GOTO n, n étant le numéro de ligne où commence le « piège » (trap) de traitement de l'erreur, ne peut être utilisée qu'une fois au cours du programme.
A l'apparition d'une erreur quelconque (détectée par le système), si le programmeur n'adopte pas l'instruction ON ERROR GOTO, le système donnera son propre diagnostic et arrêtera l'exécution du programme, même si l'erreur est minime. Le programme quittera la phase d'exécution et ne pourra la reprendre qu'à partir du début. En revanche, l'instruction ON ERROR GOTO n provoquera l'exécution d'un module de traitement des erreurs, débutant à la ligne n. En analysant le type d'erreur, le module décidera d'arrêter l'exécution, ou de la poursuivre en demandant à l'opérateur une correction opportune.
Le code numérique qualifiant le type d'erreur commise est contenu dans une variable nommée ERR, alors que le numéro de la ligne où l'on a constaté l'erreur est représenté par la variable ERL. Ces deux variables (dont le nom diffère selon le type de Basic) sont réservées au système (« variables système »).
En haut de la page 419 est présenté l'organigramme général d'implantation de ce module. L'instruction ON ERROR GOTO 1000, insérée au début du programme, mènera en ligne 1000 lorsqu'une erreur apparaîtra. Le module 1000 prévoit la correction de deux erreurs (types 26 et 41). L'apparition d'une des deux erreurs déterminera l'appel du sous-programme de traitement correspondant. Si, au contraire, l'erreur détectée n'est pas d'un type prévu, l'instruction ON ERROR GOTO 0 (qui annule ON ERROR GOTO 1000), rendra le contrôle au système ; ce dernier, après avoir émis le diagnostic Basic normal, arrêtera l'exécution. Le contrôle de bon fonctionnement des sous-programmes de traitement des erreurs peut être fait avec l'instruction ERROR (n), qui simule une erreur de type n.
Les **erreurs commises en phase de saisie des données** ne sont normalement pas détectées par le système.
Le programme d'application contrôlera lui-même l'exactitude des données saisies et, en cas d'erreur, affichera un message avant de formuler une nouvelle demande. Les seuls contrôles effectués par le système concernent le nombre et le type des variables demandées en entrée.
En phase de saisie des données, le contrôle de validité sur les valeurs fournies en entrée du programme est déterminant. Deux logiques différentes pourront être mises en œuvre. La première prévoit un contrôle immédiat, dans le sous-programme de saisie lui-même ; la seconde consiste à implanter une routine de contrôle appropriée, qui sera appelée par chaque instruction de saisie.
La première méthode sera appliquée à un contrôle non répétitif, qui reste lié à une seule phase de saisie. Sa programmation ne présente aucune difficulté : on effectuera les tests nécessaires immédiatement après la saisie. Si le contrôle s'avère négatif, on demandera une nouvelle saisie.
La seconde technique est employée pour regrouper dans un seul module toutes les instructions de diagnostic. Ce sous-programme généralisé, appelé par plusieurs points du programme, sera, au contraire, très complexe. Il faudra, en effet, envisager et traiter toutes les erreurs possibles. Malgré cela, l'adoption de cette méthode sera conseillée dans les programmes d'une certaine complexité, pour obtenir une programmation claire et structurée (diagnostics regroupés en un seul point du programme).

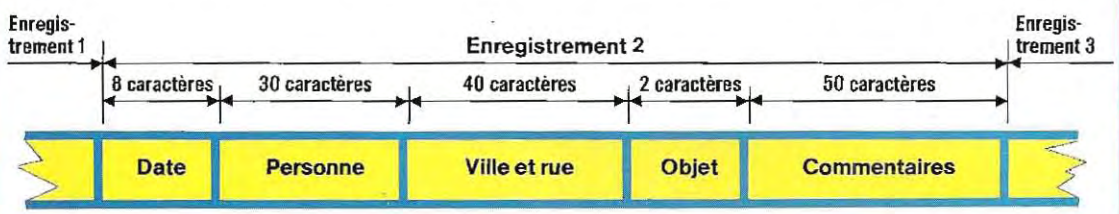
Exemple d'application = gestion d'un agenda de rendez-vous

Illustrons la mise en place de la seconde technique qui vient d'être exposée, par le programme de gestion d'un agenda de rendez-vous.
Les données sont stockées dans un fichier appelé AGENDA, ordonné chronologiquement.
Le format d'enregistrement est le suivant (voir schéma en bas de la page 419) :

DETECTION ET CORRECTION EVENTUELLE DES ERREURS



FORMAT D'ENREGISTREMENT DU FICHIER AGENDA



La « tourismatique » est née

Nul se songerait à nier l'importance décisive du rôle joué par le tourisme dans l'économie de certains pays d'Europe et son importance dans l'équilibre de leur balance des paiements.
Selon des sources officielles, on a dénombré, pour l'année 1983, 54 millions de nuits d'hôtels, sur Paris et sa région.
Pour faire face aux perspectives de crise qui semblent atteindre même le secteur touristique des pays industrialisés, pour affronter la concurrence d'autres pays particulièrement équipés dans le domaine du tourisme organisé, il a fallu améliorer la structuration des activités touristiques à l'aide de techniques adéquates parmi lesquelles l'informatique joue un rôle fondamental.

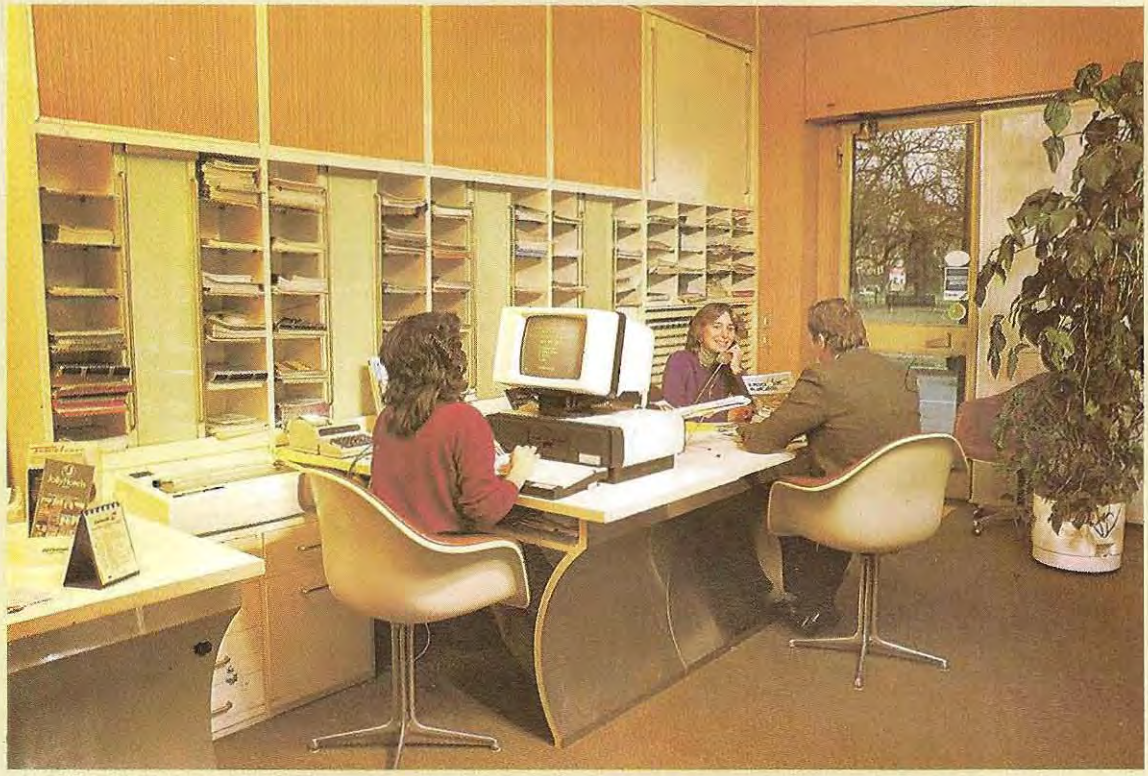
* * *

Aujourd'hui, les progrès de la technologie et la demande d'un public toujours plus exigeant ont provoqué le rapprochement de l'informatique et des télécommunications, deux mondes qui, jusqu'à présent, s'étaient développé séparément.
Dans plusieurs pays d'Europe (France, Italie, Hollande, entre autres), de nouveaux services destinés aux secteurs du tourisme et de l'hôtellerie ont été mis en place.
Ils fonctionnent en étroite collaboration avec des sociétés d'informatique qui ont fourni micro-ordinateurs et mini-ordinateurs.

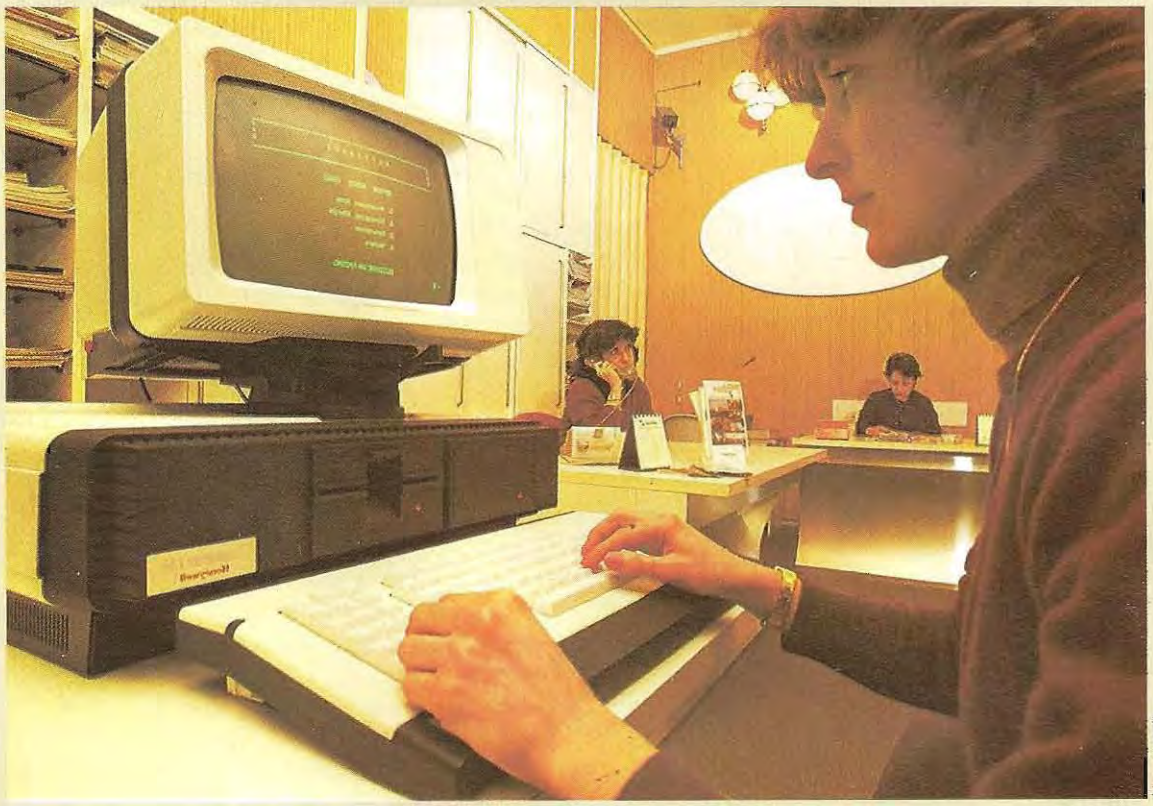
La ***gestion d'un hôtel****, par exemple, est d'une grande complexité et s'avère difficile à rationaliser.*
D'une part, il faut tenir compte d'un élément fondamental, directement issu de la tradition hôtelière, qui est la prestation du personnel, les services qu'il rend et ses qualités d'amabilité et de courtoisie. Cet élément représente encore l'un des critères du choix d'un hôtel et une garantie de sa qualité.
D'autre part, le secteur hôtelier ne peut faire abstraction aujourd'hui des problèmes et des objectifs économiques fondamentaux de notre société moderne :

– détermination du rapport qualité-prix pour l'optimisation correcte du chiffre d'affaires ;
– rationalisation de la structure d'organisation interne en vue d'une diminution des coûts.

La réalisation de ces deux objectifs est liée à l'expérience et aux capacités d'entreprise de la direction, mais elle peut être améliorée par l'utilisation d'un système de gestion automatisée des informations.
Ce système, appliqué aux différents secteurs de l'activité hôtelière (accueil, logement et restauration), offrira au personnel une base de travail plus efficace. Grâce au réseau public de transmission de données (Transpac en France), accessible à tous les usagers, de n'importe quel point géographique du pays (la distance n'ayant pas d'incidence sur la tarification), le simple particulier peut à tout moment s'informer de la disponibilité des places sur les vols de compagnies aériennes et dans les hôtels de toute catégorie, pour peu qu'ils soient équipés du matériel informatique adéquat.
L'usager peut non seulement ***s'informer****, mais aussi* ***réserver*** *une place d'avion ou une chambre d'hôtel sans se déranger, par l'intermédiaire d'un terminal ou du téléphone.*
Le système est semblable à Télétel, qui fournit, grâce à un petit terminal, des informations sur les spectacles ou le prix des voitures d'une marque donnée, mais le mode est interactif, dans la mesure où la réservation est enregistrée auprès de l'organisme (compagnie aérienne ou hôtel), grâce à l'ordinateur, sans que les personnes se soient parlé.
Les clients ont aussi à leur disposition un système de réservation automatique, qui bénéficie de toutes les possibilités de traitement de l'ordinateur. La question est, naturellement, de savoir dans quelle proportion le simple particulier est disposé (et capable) de s'équiper d'un terminal.
C'est pourquoi les ***agences de voyages*** *voient leur rôle d'intermédiaire se développer chaque jour davantage.*
Devenant elles-mêmes de véritables bureaux de tourisme, elles proposent à leurs clients des prestations de service de plus en plus sophistiquées, et pour lesquelles le support informatique se révèle plus que jamais nécessaire.
Aujourd'hui, l'agent de voyages entreprenant, et son équipe, doivent faire face à de multiples problèmes :

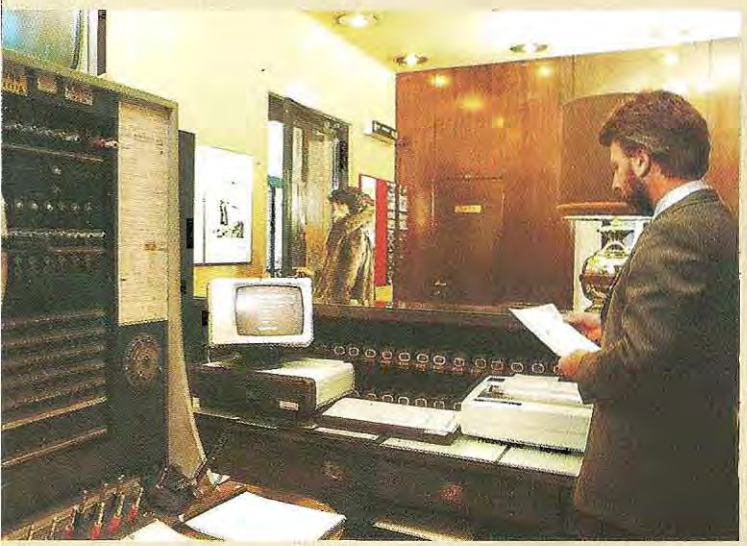


Honeywell



Honeywell

Micro-ordinateur personnel Honeywell Questar/M9050, installé dans les bureaux d'une agence de voyages.



Honeywell

Projet « tourismatique » : un micro-ordinateur Questar/M9050 à la réception d'un hôtel.

– engagement toujours plus poussé dans la mise en place de voyages organisés et donc la gestion des services fournis lors des pointes saisonnières ;
– normes de plus en plus strictes dans les domaines fiscaux et administratifs ;
– nécessité absolue d'offrir à leur clientèle un service public d'excellente qualité, passant par un personnel hautement sélectionné.

* * *

Il semblerait que l'informatisation des agences de voyages ait précédé celle des structures hôtelières. Au cours de ces dernières années, on a constaté néanmoins un fort développement de la demande de matériel et de logiciel dans le domaine de la gestion hôtelière.
Il s'agit d'une part de la gestion des comptes de l'hôtel (paie du personnel, budget, restauration, investissements éventuels pour l'entretien et l'aménagement des locaux), mais également de la gestion des réservations et des comptes clients.
L'informatisation des services hôteliers implique une restructuration du travail des diverses catégories de personnel, mais il est toutefois nécessaire que le changement se réalise dans un contexte qui valorise le capital de compétence et de professionalisme des employés.
Il faut donc comprendre en quoi consistent les nouvelles techniques d'automation et comment elles peuvent être appliquées à la réalité touristique.
Le terme ***tourismatique*** *ne doit pas faire croire qu'il suffit d'appliquer aux hôtels et aux agences de voyages des techniques informatiques déjà adoptées par d'autres secteurs. Il s'agit en fait de développer des moyens spécifiques permettant la rationalisation et l'évolution du service touristique.*

* * *

En Italie par exemple, c'est précisément ce que la société Honeywell a présenté aux professionnels du secteur touristique : deux solutions d'application sur micro-ordinateurs et mini-ordinateurs, l'une étudiée pour la gestion des hôtels, l'autre pour les agences de voyages.

La solution appliquée aux hôtels fonctionne soit sur micro-ordinateur Honeywell Questar/M, soit sur mini-ordinateur Honeywell DPS 6, chacun desservant plusieurs postes de travail et s'articulant sur les cinq modules suivants :

– réservations ;
– check-in ;
– gestion des besoins du client sur place ;
– check-out ;
– administration et comptabilité.

Le module de ***réservation*** *permet de gérer la disponibilité des chambres sur une période de deux ans, d'effectuer des réservations multiples (plusieurs chambres avec une seule réservation), de tenir compte des préférences des habitués, d'imprimer la liste des départs et des arrivées pour tous les services de l'hôtel qui en ont besoin.*
Il permet aussi la gestion de ce qu'on appelle l'over-booking (quand la demande est trop élevée) et, au contraire, en période creuse, le regroupement des réservations dans certaines parties ou étages de l'hôtel, de façon à limiter à ces parties les dépenses (chauffage et services).
Le module ***check-in*** *assure l'enregistrement des clients qui arrivent (s'il s'agit de clients réguliers, les données sont automatiquement prélevées dans les archives), la mise à jour des différents fichiers, le relevé des papiers d'identité pour la Préfecture, évitant ainsi que,*

à la suite d'un oubli ou d'une erreur, ces documents soient mal établis.
Le module de **gestion des besoins du client** sur place se charge de la mise à jour de la situation comptable du client à tout moment sur la base des services qu'il a demandés (blanchisserie, téléphone, mini-bar), prenant en compte aussi les conventions spéciales dont il pourrait bénéficier. Il prévoit, en outre, la gestion des changements de chambre éventuels et, au moment du départ, met à jour les différentes archives parmi lesquelles l'archive réservations.
Le module **check-out** sert à émettre pour chaque client, au moment de son départ, la facturation des divers mouvements de son compte. Simultanément, ce module assure la mise à jour des archives comptables de l'hôtel.
Ces dernières sont gérées par le module **administration et comptabilité de gestion**, assurant de plus l'impression des différents documents utiles :

- registre de caisse ;
- registre de débit ;
- bordereaux pour le service comptable ;
- situation et récapitulation des comptes.

D'une part, le système est caractérisé par une utilisation facile, respectant les traditions hôtelières, puisqu'il doit être utilisé par un personnel généralement dénué d'expérience en informatique et habitué à un autre type de langage et de comportement. D'autre part, il se distingue par sa grande flexibilité, particulièrement nécessaire dans un milieu où la règle est, depuis toujours, de s'adapter à des situations non standardisées. Son utilité est reconnue, par exemple, dans le cas d'arrivées ou de départs non prévus, de comptes séparés pour des occupants d'une même chambre ou, au contraire, de débit de chambres différentes sur le même compte, et enfin, de rectification de dernière minute du compte d'un client, etc.
La solution informatique proposée aux agences de voyages fonctionne sur micro-ordinateurs Honeywell Questar/M avec différents terminaux. Elle est fondée sur quatre modules :

- gestion des dossiers de voyage ;
- gestion des guichets ;
- gestion des réservations faites ailleurs.
- comptabilité générale.

Le module **de gestion des dossiers de voyage** traite aussi bien le transport que l'hébergement, ainsi que les différents services qui y sont liés, tant pour les particuliers que pour les groupes. Sont prévus tous les changements de date, calculs de quota et éventuellement, montant de devises par voyageur.
Les documents émis par le module sont : bons de réservation, confirmations de voyage, listes de voyageurs, moyens de transport, hôtels et services, listes de billets à émettre. Il est relié, après confirmation préalable de l'opérateur, au module de comptabilité générale pour l'enregistrement comptable des opérations, l'émission de factures, les extraits de compte.
Le module **gestion des guichets** concerne les billets d'avion, de chemin de fer, etc., qu'ils soient pris sur place ou ailleurs. Ce module assure une véritable gestion de magasin avec contrôle des stocks. Il gère en outre une archive billets avec mémorisation des diverses données : émission, remboursement ou annulation éventuelle du billet.
Le module de **gestion des réservations par intermédiaire** concerne les opérations de réservation effectuées par un client auprès d'une autre compagnie (nationale ou étrangère). Il assure l'émission de bons de réservation analogues à ceux émis par le module dossiers de voyage (ces derniers formant une seule archive « bons de réservation » pour les contrôles d'exploitation et de comptabilité ultérieurs), ainsi que l'émission de documents propres à la comptabilité de l'intermédiaire lui-même (commission sur ventes, etc.).
Enfin, le module de **comptabilité générale**, au moyen d'un plan de compte personnalisé, gère la comptabilité clients-fournisseurs à comptes ouverts, permet l'enregistrement de la T.V.A., comptabilise les coûts pour les services comptables (dossiers de voyage), effectue les bilans de vérification, de récapitulation et de clôtures périodiques.
Comme nous l'avons déjà dit, la majeure partie des mouvements comptables étant automatiquement exécutée par les autres modules, elle permet une réduction maximale du travail manuel et limite la possibilité d'erreurs.

1 / Dates des rendez-vous :
Mois = 2 caractères (entier, de 1 à 12).
Jour = 2 caractères (entier, de 1 à 31).
Heure = 2 caractères (entier, de 8 à 20 ; seules sont considérées les heures diurnes).
Minutes = 2 caractères (entier, de 0 à 50 par pas de 10).
On arrondira systématiquement les minutes à la dizaine supérieure.

2 / Personne à rencontrer : 30 caractères (Nom et Prénom).

3 / Ville et rue : 40 caractères.

4 / Objet du rendez-vous : 2 caractères (code).

5 / Commentaires : 50 caractères.

La longueur du fichier sera mémorisée dans l'enregistrement 1.
Le programme aura à exécuter les fonctions suivantes :

a) impression de la liste des rendez-vous entre deux dates fixées ;
b) sélection des rendez-vous se passant dans une ville déterminée ;
c) sélection des rendez-vous selon un objet précis.

Un tel programme s'articulera ainsi :

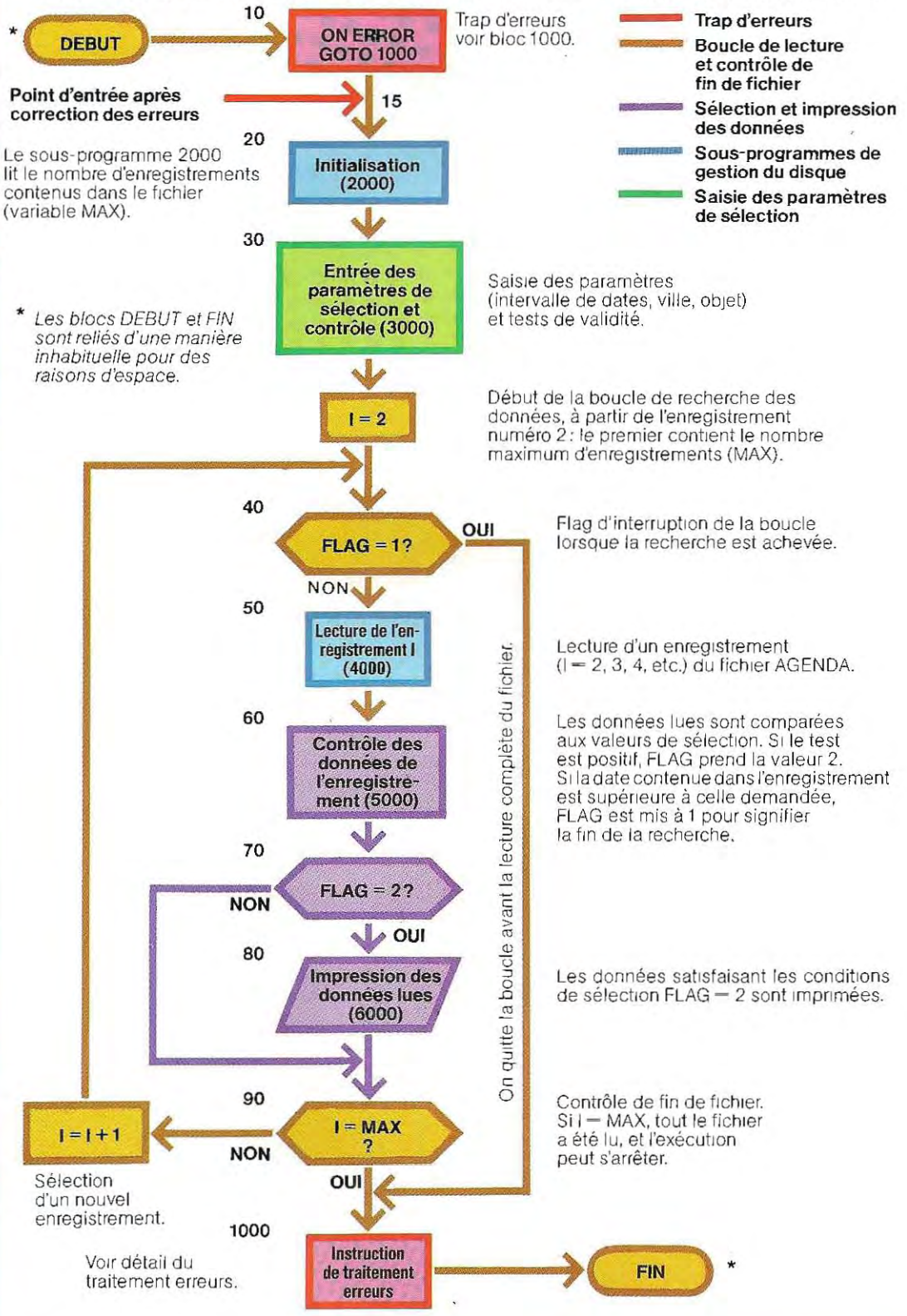
- initialisation ;
- lecture des paramètres de sélection ;
- recherche et impression des données.

En phase d'initialisation, on vérifiera l'existence du fichier AGENDA, et on lira sa longueur dans l'enregistrement 1 (voir p. 231).
La lecture des paramètres de sélection consistera concrètement à saisir l'intervalle de dates, la ville ou l'objet souhaités (ces deux derniers paramètres étant optionnels).
La dernière partie prévoit la lecture séquentielle du fichier et, pour chacun des enregistrements, la vérification de sa correspondance avec les données saisies. Si les critères de recherche sont satisfaits, on imprimera l'enregistrement avant de traiter le suivant.
La page 425 présente l'organigramme de base : un indicateur a été prévu pour sortir de la boucle avant la lecture complète du fichier, dans le cas où la date contenue dans l'enregistrement lu est supérieure à celle demandée (FLAG = 1).
Le parcours du fichier met en œuvre la technique des boucles à rupture de séquence : le fichier est lu séquentiellement, à partir du premier enregistrement. Si la date contenue dans l'enregistrement lu est inférieure à la valeur maximale saisie, il faudra continuer la

Avec un micro-ordinateur, la gestion d'un agenda devient une simple formalité.

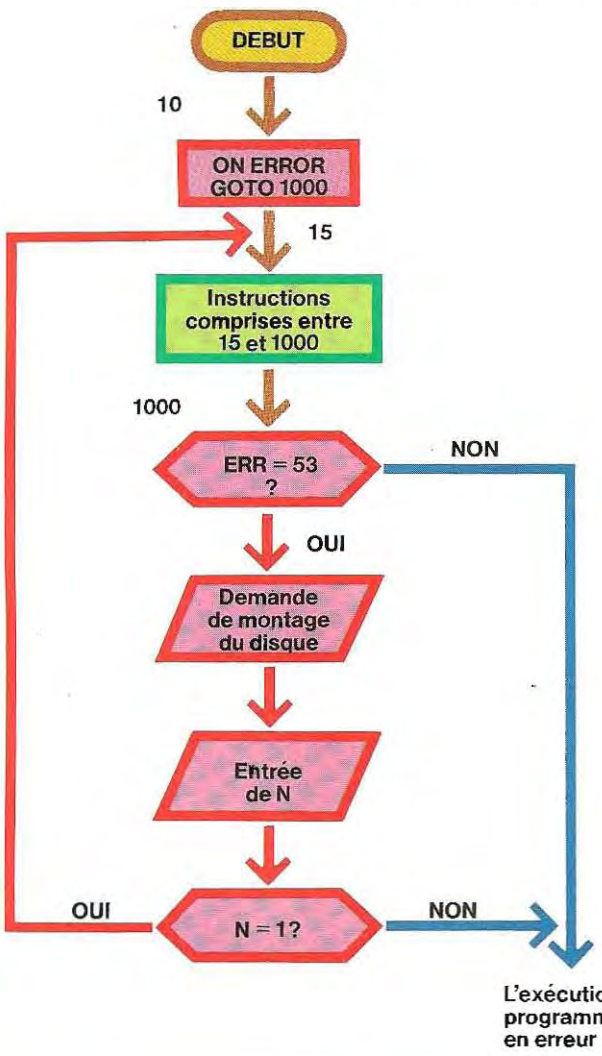
Honeywell

ORGANIGRAMME SIMPLIFIE DU PROGRAMME DE GESTION AGENDA



lecture afin de trouver les autres enregistrements éventuels à sélectionner. La recherche peut être abandonnée au premier enregistrement contenant une date supérieure à la date maximale saisie.
Ci-dessous, on a détaillé le schéma du module de traitement des erreurs (bloc 1000). En ce cas, la seule erreur système correspond à l'absence du fichier AGENDA sur disque (code 53, à titre d'exemple). A l'apparition de cette erreur, on affiche un message d'avertissement; la machine attendra ensuite la fin des opérations de correction que l'utilisateur doit effectuer (en ce cas particulier le remplacement de la disquette). On suspendra l'exécution en insérant une instruction d'entrée à la console; l'opérateur disposera alors de tout le temps nécessaire pour remplacer le disque; en tapant la valeur choisie (ici 1), il réactivera le traitement.
L'organigramme de la page 427 détaille la saisie des paramètres de sélection (routine 3000); alors que le suivant (page 428) correspond au sous-programme 5000, prévu pour le traitement des données lues dans le fichier, en fonction des paramètres établis.
Le sous-programme 5000 lira les données MOIS, JOUR, HEURE, MINUTE, VILLE, OBJET, de chaque enregistrement du fichier AGENDA, après avoir reçu en entrée les para-

DETAIL DU MODULE DE TRAITEMENT DES ERREURS



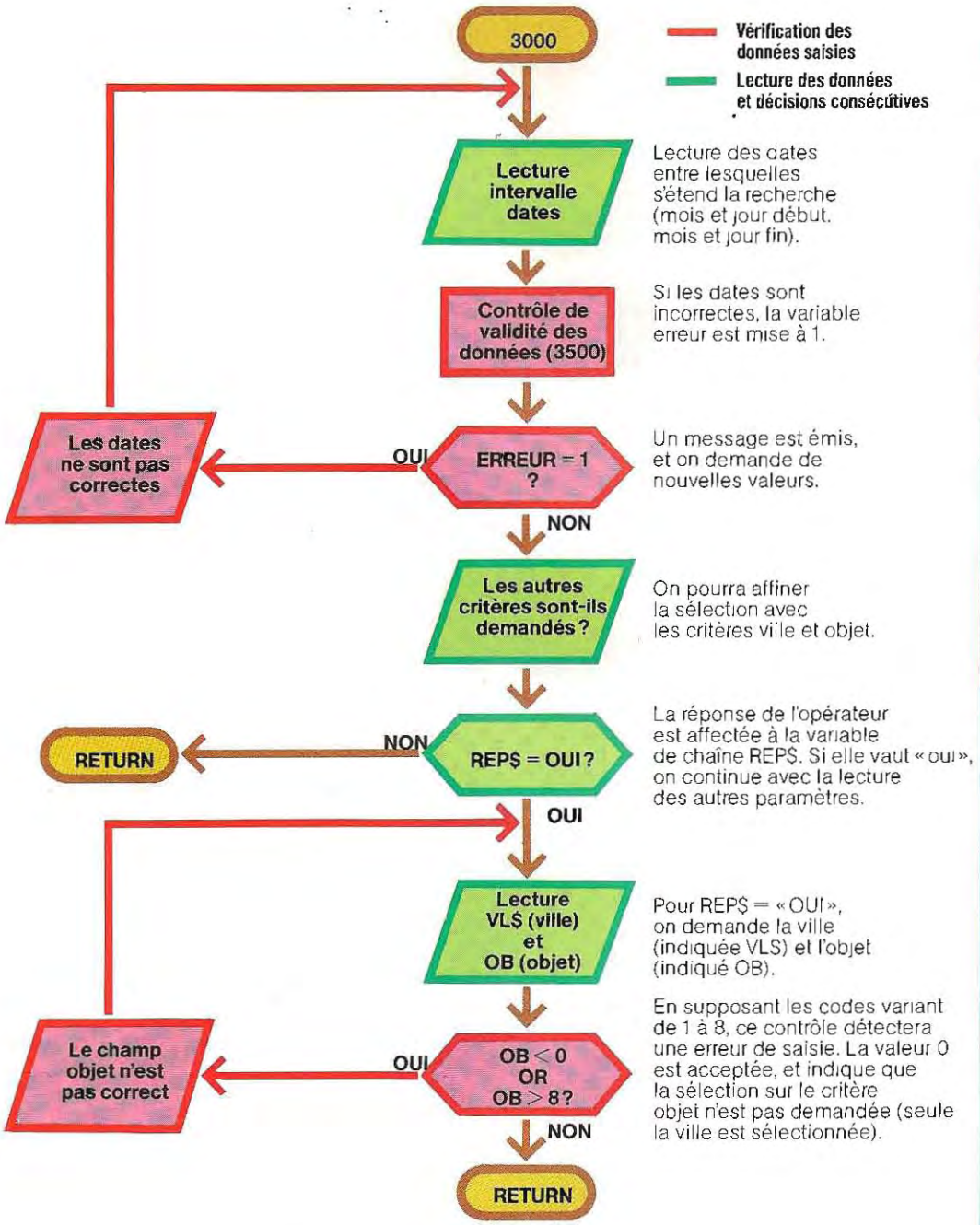
Voir organigramme simplifié du programme de gestion du fichier AGENDA.

Erreur de code 53 : le fichier AGENDA n'a pas été trouvé sur le disque. Autre erreur : pas de traitement prévu.

Affichage d'un message d'avertissement.

Pour donner à l'opérateur le temps de monter le disque, le système se met en attente sur la saisie d'une variante quelconque (par ex., N). L'opérateur peut vérifier le disque et décidera de continuer (en entrant la valeur 1) ou d'arrêter l'exécution (en entrant une valeur quelconque différente de 1).

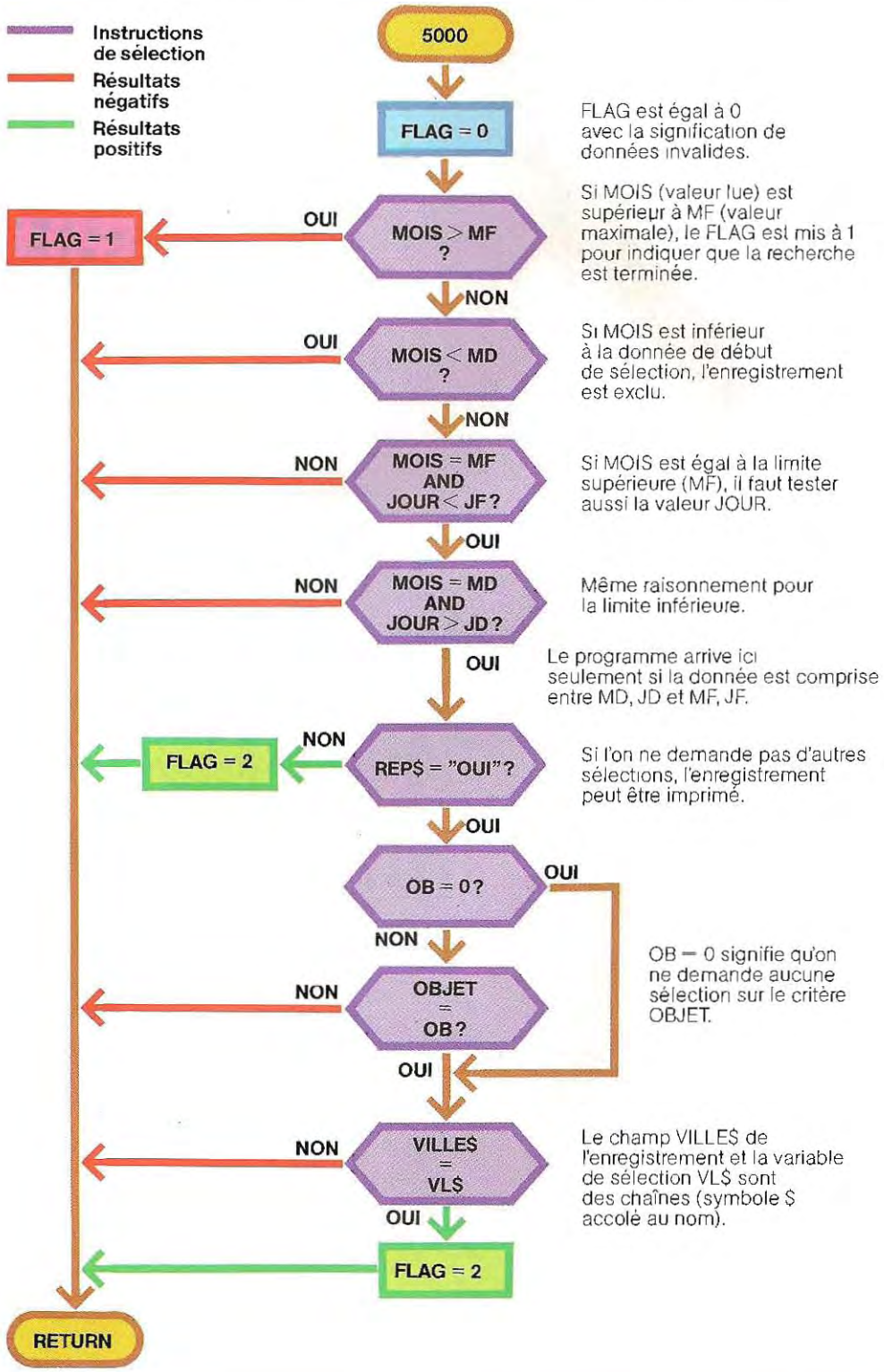
SOUS-PROGRAMME DE SAISIE DES PARAMETRES DE SELECTION



Paramètres fournis en sortie :

JD (jour), MD (mois) = date de début de sélection
JF (jour), MF (mois) = date de fin de sélection
REP$ = "OUI" si d'autres sélections sont demandées
VL$ = ville à sélectionner (chaînes de caractères)
OB = objet (code)

SOUS-PROGRAMME DE TRAITEMENT DES DONNEES



mètres (date de début de la recherche, date de fin de la recherche, ville, objet) acquis par le sous-programme 3000.
Dans le sous-programme 5000, la comparaison entre les deux chaînes VILLE$ et VL$ se résume à un simple test :

(IF VILLE$ = VL$ GOTO ...).
En sortie du sous-programme 5000, le paramètre FLAG prendra les valeurs :

0 = enregistrement non sélectionné ;
1 = date lue supérieure à la date maxi-

PROGRAMME DE GESTION D'UN AGENDA

```
]LIST

1 REM **********************
2 REM ** PROGRAMME AGENDA  **
3 REM **********************
4 REM
5 REM
6 REM programme execute sur APPLE II
7 REM en Basic Applesoft
8 REM ** PROGRAMME PRINCIPAL **
9 REM
10 ONERR GOTO 1000
11 REM
20 GOSUB 2000: REM lecture de la taille du fichier AGENDA
21 REM
30 GOSUB 3000: REM saisie des parametres de selection
31 REM
40 FOR I = 2 TO MAX: REM debut de boucle
41 REM
50 IF FLAG = 1 GOTO 1070: REM on abandonne la lecture
51 REM
60 GOSUB 4000: REM lecture enregistrement
70 GOSUB 5000: REM controle des donnees
71 REM
80 IF FLAG < > 2 GOTO 100
81 REM
90 GOSUB 6000: REM impression des donnees selectionnees
91 REM
100 REM
101 REM
102 REM
103 REM
104 REM ** traitement des erreurs **
105 REM
1000 IF ERR < > 53 THEN PRINT "erreur non prevue": STOP
1001 REM
1010 PRINT "erreur disque"
1020 PRINT "montez la disquette contenant le fichier AGENDA"

1030 PRINT "tapez 1 pour continuer, 2 pour arreter"
1031 REM
1040 INPUT N
1050 IF N = 1 GOTO 20
1051 REM
1060 PRINT "sortie en erreur": STOP

1061 REM
1070 END
1071 REM
1072 REM
1073 REM * routine de test et lecture longueur d'AGENDA *
1074 REM (detaillee par la suite)
2000 MAX = 5: REM valeur fixe pour simulation
2010 RETURN
2011 REM
2012 REM
2013 REM
2014 REM * routine de saisie des parametres *
2015 REM
3000 PRINT "dates limites ?"
3010 INPUT "mois debut ";MD: INPUT "jour debut ";JD
3020 INPUT "mois fin ";MF: INPUT "jour fin ";JF
3030 REM controle des dates saisies
3040 IF MD > 12 OR MF > 12 OR JD > 31 OR JF > 31 THEN PRINT "erreur date": GOTO 3000
3050 INPUT "autre selection? (oui/non) ";REP$
3060 IF REP$ = "NON" GOTO 3100
3070 INPUT "ville de selection ";VL$
3090 INPUT "code objet, entre 1 et 8 ";OB
3090 IF OB < 0 OR OB > 8 THEN PRINT "code errone": GOTO 3090
3100 RETURN
3101 REM
3102 REM
3103 REM
```

```
3104 REM *routine de lecture des enregistrements*
3105 REM detaillee par la suite, elle est ici simulee
3107 REM
4000 PRINT "simulation de lecture sur disque"
4020 INPUT "mois ";MOIS: INPUT "jour ";JOUR
4030 INPUT "ville ";VILL$: INPUT "objet ";OBJET
4040 RETURN
4041 REM
4042 REM
4044 REM *routine de selection*

4045 REM
5000 FLAG = 0
5010 MO$ = "date hors des limites de selection"
5011 REM
5020 IF (MOIS > MF) OR (MOIS = MF AND JOUR > JF) THEN FLAG = 1: PRINT MO$: GOTO 5090
5030 IF (MOIS < MD) OR (MOIS = MD AND JOUR < JD) THEN PRINT MO$: GOTO 5090
5031 REM
5040 IF REP$ < > "OUI" GOTO 5060
5050 IF OB = 0 GOTO 5070
5060 IF OB < > OBJET THEN PRINT "objet non selectionne": GOTO 5090
5070 IF VILL$ < > VL$ THEN PRINT "ville non selectionnee": GOTO 5090
5080 FLAG = 2
5090 RETURN
5091 REM
5092 REM
5094 REM *routine d'impression des resultats*
5095 REM
6000 PRINT "RENDEZ-VOUS PREVUS"
6010 PRINT "DU : ";JD;"/";MD;" AU : ";JF;"/";MF
6020 PRINT "donnees memorisees"
6030 PRINT "mois ";MOIS
6040 PRINT "jour ";JOUR
6050 PRINT "objet ";OBJET
6060 PRINT "ville ";VILL$
6080 RETURN

]RUN
dates limites?
mois debut 3
jour debut 14
mois fin 5
jour fin 16
autre selection? (oui/non) OUI
ville de selection PARIS
code objet, entre 1 et 3 3
simulation de lecture sur disque
mois 4
jour 12
ville LYON
objet 3
ville non selectionnee
simulation de lecture sur disque
mois 3
jour 22
ville PARIS
objet 3
RENDEZ-VOUS PREVUS
DU : 14/3 AU : 16/5
donnees memorisees
mois 3
jour 22
objet 3
ville PARIS
simulation de lecture sur disque
mois 5
jour 1
ville LYON
objet 2
date hors des limites de selection
```

male choisie; il faut dorénavant cesser la recherche;

2 = sélection réussie; impression des données de l'enregistrement.

Le listing de la page 429 réunit les différents modules :
programme principal : organigramme de la page 425;
sous-programme 3000 : organigramme de la page 427;
sous-programme 5000 : organigramme de la page 428,
ainsi qu'un exemple d'exécution. Les parties manquantes du programme (modules de gestion du disque et de lecture des enregistrements), seront présentées par la suite. Pour l'instant, quelques instructions les simuleront.

En ajoutant une routine de saisie des données, nous aurons ainsi constitué un programme complet de gestion d'un agenda.

L'homme face à la machine

Depuis l'avènement de la société industrialisée, l'homme n'a cessé de s'adapter aux machines qui se faisaient de plus en plus nombreuses autour de lui. De nos jours, la mode est à l'adaptation inverse, et l'on accorde plus d'attention à la «machine humaine».

*Une science se charge des problèmes posés par cette nouvelle préoccupation : c'est l'**ergonomie**, ou «l'art et la science de concevoir des systèmes capables de supprimer ou de limiter les sensations douloureuses, physiques ou mentales, réelles ou imaginaires».*

L'ergonomie étudie les conditions de confort, de santé, de sécurité et d'équilibre mental des personnes vivant dans un milieu de travail, et considère le rapport de l'homme et de la machine comme un problème à résoudre par des moyens physiologiques, psychologiques ou techniques. Si les premières tentatives d'automatisation du travail de bureau ont été l'objet d'une certaine résistance, c'est que les rapports de l'homme avec la machine ont été initialement considérés comme antagoniques. On pensait, en effet, qu'il appartenait aux opérateurs de s'adapter à ces bureaux climatisés, extrêmement soignés, certes, mais également dépourvus de vie. Cette conception représentait l'obstacle le plus important à surmonter : l'univers de la machine était imposé à l'homme.

C'est seulement depuis peu que, sous l'influence de la recherche ergonomique, les conditions de travail commencent à se modifier, dans les bureaux du moins.

La technologie de pointe n'est plus autant perçue aujourd'hui comme l'ennemi à abattre. On comprend de mieux en mieux que les machines dotées de capacités de plus en plus performantes sont destinées à faciliter le travail de l'homme. La plupart des problèmes physiques ont reçu une solution directe et immédiate dans l'élaboration électronique. Les angles des machines de bureau se sont arrondis, devenant ainsi moins dangereux.

Des études précises ont été réalisées afin de mettre en évidence la réaction de la rétine lorsque l'œil fixe un écran lumineux pendant plusieurs heures.

Le NIOSH (National Institute of Occupation Safety and Health) et l'Académie américaine pour les Sciences sont arrivés à la conclusion que la quantité de radiations émise par les écrans est trop faible pour provoquer des cataractes, ou pire, des dommages génétiques. Elles n'ont aucune répercussion sur la santé. En revanche, d'autres problèmes n'ont pas été aussi faciles à résoudre. Une enquête, également menée par le NIOSH, a recueilli quantité de données sur différents troubles réels ou imaginaires.

La cause n'est pas toujours et nécessairement imputable à la machine elle-même : beaucoup de ces troubles peuvent être provoqués par un mauvais éclairage, un siège mal adapté ou une ambiance psychologiquement désagréable. En effet, la solution d'un problème lié à la machine révèle souvent l'existence d'autres problèmes, d'origine psychologique, ceux-là.

James Martin, auteur de «Man – Computer Dialogues», affirme que chaque utilisateur nécessite une interface homme – machine différente. Les hommes appartiennent, selon lui, à des catégories hétérogènes, incapables parfois de «communiquer» entre eux et qui, devant un ordinateur, peuvent passer à l'improviste d'une catégorie à l'autre.

Martin les divise en couples : attentif - distrait, expert - inexpert, intelligent - non intelligent, actif - passif, affirmant que pour chaque utilisateur, il est nécessaire de trouver un moyen idéal de communication : alphanumérique, sonore, ou graphique. Martin écarte immédiatement le langage naturel, celui-ci demandant un logiciel trop complexe ; sa sympathie ne va pas non plus aux langages spécifiques de programmation tels le Cobol et le Fortran, ceux-ci exigeant des niveaux de professionnalisme et de motivation trop élevés pour la masse des utilisateurs.

Dans un article publié en 1982, Paul Heckel a proposé à son tour une classification d'utilisateurs d'ordinateurs, en soutenant qu'un bon concepteur de logiciel doit connaître à fond son public.

«Tout le monde parle sa propre langue, mais chacun de nous a ses propres expressions et, selon sa profession, un jargon lié à une activité donnée. De ce fait, des individus se consacrant à des professions différentes tendront à penser aussi d'une manière différente. Combien de fois nous a-t-on demandé de faire des programmes utilisables par tous,

même par des individus sans aucune connaissance en la matière. Si seulement cela était possible ! La plupart de nos problèmes proviennent au contraire de ce que l'on a implicitement pensé que l'utilisateur pourrait se défaire facilement des connaissances acquises auparavant dans l'exercice de sa profession et s'adonner spontanément à un nouveau jargon ».

Le jargon lui-même, cauchemar des puristes, peut faciliter l'approche avec le nouvel usager, à condition que l'on emploie avec lui le jargon qui lui est familier. Il faut parler avec la secrétaire le langage de la secrétaire, avec le médecin, le langage du médecin, avec le commerçant, le langage du commerçant. Faute de quoi l'informaticien ne saura pas cerner clairement le problème précis auquel le traitement informatique doit être appliqué. Ce qui est vrai du logiciel l'est aussi du matériel.

Un manuel, une page vide ou un menu mal écrit peuvent nuire à l'esprit, davantage qu'une mauvaise adaptation du clavier à la main. Un curseur qui clignote peut, seulement en théorie, attendre éternellement qu'on appuie sur la touche requise, car, pour l'esprit humain, le problème se pose en d'autres termes. Selon « Convergent Technologies », une société de projets ergonomiques, un opérateur ne peut attendre une réponse pendant plus d'une seconde et demie. En revanche, dans « Man – Computeur Dialogues », Martin affirme que si le temps de réponse est trop bref, l'utilisateur inexpérimenté suspecte une inexactitude des données. Le fait que l'attention humaine n'est soutenue que pendant un laps de temps limité constitue de toute manière un des aspects les plus critiques du face à face homme-machine. Ces derniers temps, on a longuement discuté de l'opportunité de l'utilisation de ***menus*** *au lieu de commandes. Avec les menus, l'utilisateur n'a qu'à choisir parmi un certain nombre de propositions, sans avoir à se souvenir de toutes les commandes, l'ordinateur lui présentant automatiquement une liste de choix possibles. De cette manière, le temps passé à apprendre est considérablement réduit puisqu'aucun effort de mémoire n'est plus nécessaire. Très appréciés des débutants, les menus sont terriblement ennuyeux pour les utilisateurs expérimentés, obligés de subir, écran après écran, des choix qu'ils connaissent par cœur. C'est pour cette raison qu'un bon « package »* doit avoir plusieurs sous-programmes de « by-pass », offrant des raccourcis aux utilisateurs plus expérimentés. Même les jeux vidéo de type Dragons et Donjons tiennent compte de ce principe.*

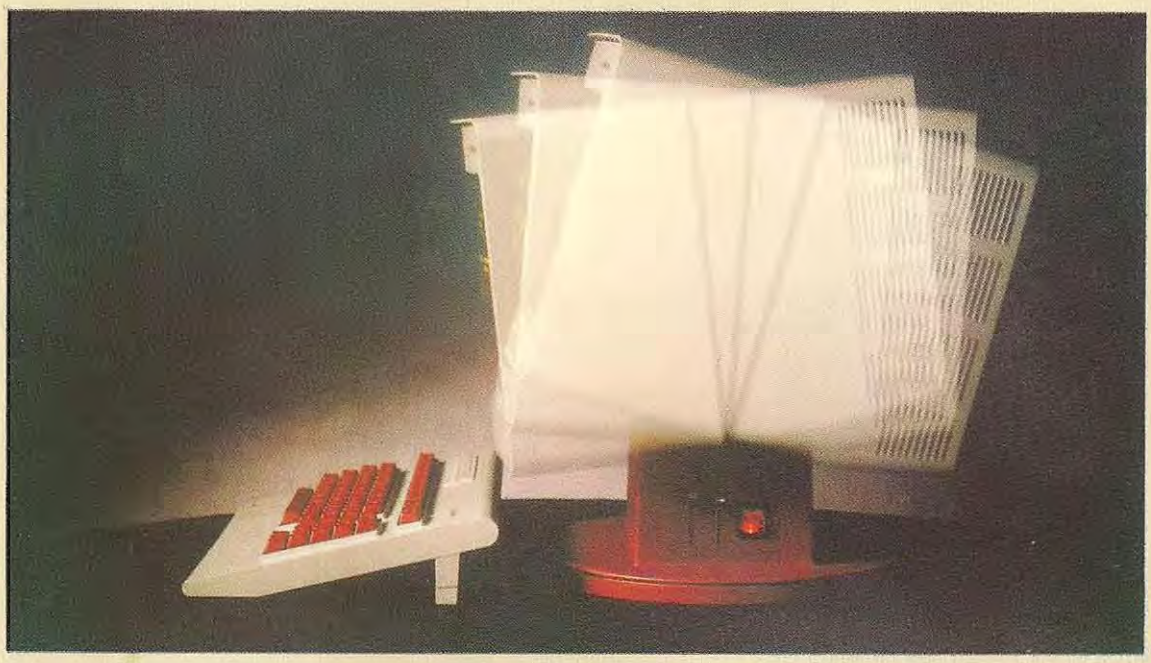
Quand on parle avec un inconnu, généralement on cherche d'instinct à évaluer son niveau intellectuel : comprend-il ce qu'on lui dit ? Participe-t-il à la discussion ? De cette manière, on se fait une opinion qui pousse à adopter une certaine intonation et un certain type de langage. Pourquoi donc un logiciel ne devrait-il pas réagir de la même manière ? Selon Gerry Gassman, responsable d'ergonomie des systèmes de gestion de Hewlett-Packard, on doit surtout définir avec le plus grand soin le contexte dans lequel la nouvelle machine devra être utilisée, chaque milieu nécessitant une approche différente.

Par exemple, dans un centre informatique ou dans une zone de service, les opérateurs travaillent le plus souvent debout.

Disposition ergonomique des terminaux d'un centre de calcul de la Société Hewlett-Packard aux Etats-Unis.

K. Reese/Marka

* Les « packages » sont des ensembles de programmes d'application, destinés à la solution de problèmes scientifiques ou de gestion et mis en vente en complément des systèmes de base.



Ericsson

L'écran orientable est un élément de confort appréciable.

Dans ce cas, le tableau de commande sera plus visible et plus facile à atteindre s'il se trouve sur le devant et vers le haut de l'appareil. Au contraire, si l'utilisateur travaille assis à un bureau, les commandes seront à hauteur des yeux ; de même le pupitre de commande et les unités d'E/S devront être à portée de la main. Une soigneuse analyse des modalités d'utilisation accompagne toujours la recherche de ceux qui conçoivent la forme des ordinateurs. Confiée à des spécialistes d'esthétique et de psychologie industrielles, cette recherche particulière implique des entretiens avec des utilisateurs potentiels, la construction de prototypes et l'essai de ces machines en particulier par des utilisateurs néophytes. Selon M. Gassman, la société Hewlett-Packard s'impose des contraintes encore plus sévères que celles établies par la loi, afin de pouvoir être toujours à l'avant-garde et de ne jamais être prise au dépourvu par une nouvelle norme.

De nombreux problèmes ergonomiques ont été résolus sur le plan technique, mais il reste encore beaucoup à faire en pratique. Tandis qu'aux Etats-Unis on se fie essentiellement à la loi du marché, avec la certitude qu'il est plus facile de vendre des machines à la fois belles et pratiques, les Européens sont plutôt enclins à légiférer. C'est ainsi qu'en octobre 1980, l'Institut d'assurances contre les accidents du travail d'Allemagne Fédérale a publié une liste de normes de sécurité concernant le matériel de bureau. Depuis janvier 1982, la RFA est le premier pays au monde à disposer d'une ***législation*** *en matière d'ergonomie.*

A la CBEMA (Computer and Business Equipment Manufacturers Association), où l'on étudie ce problème dans une optique mondiale, on reconnaît la nécessité d'une ***normalisation*** *tout en émettant des réserves sur la validité technique et la souplesse des standards de référence. En effet, dans le monde entier, l'ergonomie se contente, le plus souvent, de corriger des erreurs de conception parce qu'elles ont été signalées comme responsables de troubles physiques chez l'utilisateur. L'ergonomie ne se limite pourtant pas à cela, et de nombreux spécialistes cherchent à construire des appareils capables, par leur forme, de mettre l'utilisateur dans les meilleures conditions matérielles susceptibles d'encourager leur créativité.*

On a ainsi pu élaborer une nouvelle conception de l'ergonomie qui ferait de la machine un véritable stimulateur de l'activité cérébrale humaine.

(d'après DATA MANAGER, septembre 1983 ; © INTERACT janvier-février 1983).

Les fonctions

On appelle fonction une relation qui associe une valeur à une ou plusieurs autres.
Le cas le plus simple et que l'on rencontre le plus fréquemment est celui de fonctions qui ne relient que deux variables et qui, à une valeur de l'une, font correspondre une valeur de l'autre.
On peut, par exemple, dire que la distance parcourue par une voiture avec un litre de carburant est fonction de sa vitesse.
Pour déterminer la relation qui associe le trajet accompli à la vitesse, il nous faudra mettre dans le réservoir de la voiture une quantité donnée de carburant (1 litre par exemple) et mesurer le nombre de kilomètres que nous réussirons à parcourir avec ce carburant en maintenant une vitesse constante (ce paramètre doit être fixe également).

En répétant cette expérience avec des vitesses différentes (toujours avec la même quantité d'essence) et en mesurant chaque fois le parcours effectué, on obtiendra une table de correspondance entre la vitesse et la distance couverte :

Vitesse km/h	Distance parcourue (km pour un litre)
40	16
60	14
80	12
100	11
120	10

Ce tableau donne une représentation de la fonction reliant le trajet à la vitesse, mais ne permet pas de déterminer toutes les caractéristiques de cette relation.

Vade-mecum de la console idéale

Ce sont sans doute les terminaux (clavier et écran) qui ont bénéficié des études ergonomiques les plus poussées. Voici les critères de qualité qui reviennent le plus souvent :

1. Les opérateurs sont amenés à changer fréquemment de position durant leur travail face au terminal. Après plusieurs heures de saisie de données et de traitement de texte ou d'une activité comparable, les épaules ont tendance à s'affaisser et la tête à s'incliner d'une vingtaine de degrés par rapport à la verticale. Pour tenir compte du fait que les gens ne restent pas figés dans une position immuable, l'écran devrait former avec l'horizontale un angle moyen situé entre 10 et 40°. En outre, il doit être orientable en fonction d'éventuelles variations d'éclairage.

2. L'écran devrait être placé à une cinquantaine de centimètres des yeux de l'utilisateur ; le clavier et le pupitre aussi. Des changements continuels de longueur focale provoquent en effet une fatigue rapide de la vue.
Sur un système vraiment bien conçu, l'opérateur devrait pouvoir ne régler qu'un de ses paramètres, les autres s'adaptant automatiquement.

3. Toutes les touches de commande doivent être à portée de main, le bras étant demi-fléchi.

4. Les ventilateurs qui assurent le refroidissement des terminaux et des unités de disque risquent de provoquer divers troubles (dessèchement de la cornée, douleurs rhumatismales). Les courants d'air chaud devront, par conséquent, être détournés du visage et ceux d'air froid du reste du corps.

5. L'opérateur doit pouvoir s'installer correctement. Dans ce but, un pupitre à surface anti-reflet prendra place derrière le clavier. Une inclinaison réglable entre 15 et 75° par rapport à la verticale permettra d'y écrire plus facilement.

6. Les écrans à caractères verts ou jaunes semblent les moins fatiguants pour la vue. Si les premiers sont adaptés aux éclairages puissants, les seconds conviennent mieux aux lumières diffuses. Le choix dépend donc non seulement des goûts de chacun mais des caractéristiques du bureau.

7. Selon certains experts, les passages perpétuels d'un texte écrit noir sur blanc à l'écran où le contraste est inversé fatiguerait les pupilles en les obligeant à s'adapter continuellement. Pour d'autres, la lumière émise par l'écran n'a pas le même effet sur l'œil que celle réfléchie par le papier ; la rétine aurait tendance à atténuer le contraste en créant une sorte de halo autour des caractères lumineux.

8. Sur l'écran, les caractères perdent peu à peu leur netteté avant d'être « rafraîchis ». L'effet stroboscopique qui en résulte est particulièrement éprouvant tant pour la vue que pour les nerfs. L'augmentation de la luminosité totale ne fait qu'accentuer ce phénomène. C'est pourquoi la plupart des écrans présentent des caractères clairs sur fond sombre et les réécrivent de 50 à 60 fois à la seconde.

9. Une réflexion importante peut rendre l'écran illisible et donner mal à la tête. On peut parer à cela en utilisant du verre dépoli, des filtres, des revêtements, ou en rendant l'écran réglable. Le verre gravé et les filtres sont néfastes à la netteté et les revêtements coûtent cher. L'adoption d'un écran réglable constitue, en revanche, une solution simple et économique.

Définition d'un diagramme par points

Pour visualiser la fonction plus aisément et de façon plus détaillée, on peut établir un graphique (le diagramme par points de la fonction) à l'aide du tableau que nous venons de voir.
Il convient tout d'abord de fixer une échelle pour la vitesse et une pour la distance.
Par exemple :
1 cm pour 20 km
5 mm pour 2 km parcourus avec un litre de carburant.
Suivant cette convention, la vitesse de 40 km/h sera représentée par un segment de 2 cm et le parcours correspondant (16 km) par un segment de 40 mm.
Pour construire le graphique, nous devons transcrire les valeurs de la table en cm pour les vitesses et en mm pour les distances, selon l'échelle utilisée :

Vitesse km/h	Equivalence en cm	Distance km	Equivalence en mm
40	2	16	40
60	3	14	35
80	4	12	30
100	5	11	27,5
120	6	10	25

Désignons par **y** la valeur du parcours exprimée en mm et par **x** la valeur de la vitesse correspondante (en cm). On peut alors reporter les valeurs de **x** à l'horizontale et celles de **y** à la verticale, et établir ainsi le schéma de la page 436. On appelle axe des **x** (ou des **abscisses**) la droite horizontale supportant l'échelle des vitesses et axe des **y** (ou des **ordonnées**) la droite verticale symbolisant l'échelle des distances parcourues.
On obtient ainsi une représentation graphique de la relation entre la vitesse et la dis-

10. Le contraste est également un point important. Certaines normes européennes préconisent un taux compris entre 3 et 15. De plus, un réglage du contraste doit être possible pour l'adapter aux conditions d'éclairage et aux besoins de chacun.

11. Travailler longtemps sur un clavier est moins fatigant si l'on peut tenir les coudes pliés à 90° et si les poignets forment un angle inférieur à 10° de l'horizontale. La position du clavier devra donc le permettre.

12. Selon certains techniciens (allemands notamment), l'épaisseur du clavier idéal dépasse à peine 3 cm. Des claviers aussi minces sont fort peu répandus mais l'installation d'un appuie-main d'une certaine épaisseur devant le clavier permet de se replacer dans les conditions idéales.

13. L'inclinaison du clavier contribue également au confort d'utilisation en réduisant la fatigue des bras, des poignets et des coudes. Là aussi, les avis sont partagés : on recommande, en Allemagne, une inclinaison inférieure à 15°. D'autres assurent qu'un angle inférieur à 7° est à la fois source d'erreurs et d'inconfort.

14. Opérateurs expérimentés et novices n'ont pas toujours les mêmes besoins et il est pénible d'être placé face à l'écran, alors que l'on porte plutôt son attention sur le clavier. De même, un débutant préférera entendre le déclic de chaque touche.
Si les opérateurs chevronnés s'accommodent mieux d'un clavier moins sensible, tous s'accordent cependant sur la nécessité de percevoir une réaction minimale de la touche afin de ne pas manquer et de ne pas répéter une frappe.

15. Le « rollover » est la faculté de la machine à accepter une cadence de frappe très rapide. Des pointes de vitesse de 200 à 300 caractères/minute peuvent en effet survenir lors de la frappe de mots ou passages familiers à l'opérateur, et selon la « virtuosité » dont il est capable.

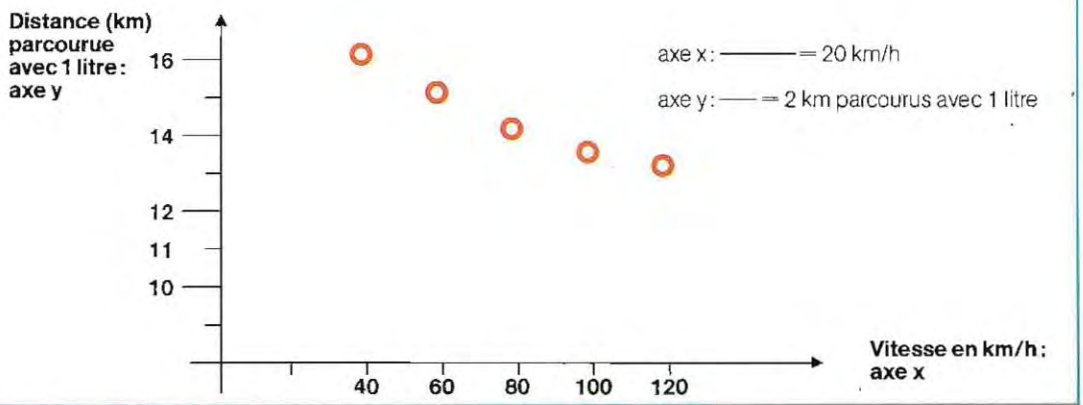
16. La disposition du clavier IBM Selectric fait désormais figure de standard et une configuration différente ne réussit qu'à troubler l'opérateur et à le retarder dans son travail. En outre, il lui faudra des semaines pour s'adapter à ce matériel. Par contre, la touche de majuscules, conçue à l'origine pour les téléscripteurs, présente peu d'utilité en informatique et serait mieux remplacée par une touche de blocage.

17. Un clavier est composé de trois zones. La zone principale comprend les caractères alphanumériques et quelques commandes d'utilisation fréquente. Les commandes d'écran sont regroupées dans une autre zone qui doit être accessible sans détourner les yeux. Quant à la troisième zone, elle nécessite un déplacement du regard, des bras et des mains et ne contrôle donc que des fonctions secondaires moins utilisées. Cette répartition varie bien entendu selon la spécialisation de l'appareil. Un opérateur de traitement de texte n'a pas les mêmes besoins qu'un comptable qui utilisera surtout le pavé numérique. En outre, l'agencement du clavier tient compte, comme celui d'une machine à écrire, des lettres les plus fréquemment employées dans une langue donnée et les répartit en conséquence.

18. Enfin, toujours dans le souci de ménager la vue, certains experts recommandent pour le clavier un coefficient de réflexion compris entre 20 et 50 pour cent.

(d'après DATA MANAGER n° 26, septembre 1983).

DIAGRAMME PAR POINTS DE LA FONCTION DISTANCE-VITESSE



tance en posant :
1 cm = 20 km/h et 5 mm = 2 km.
Ces deux valeurs (1 cm et 5 mm) sont les unités respectives de l'axe **x** et de l'axe **y** qui permettent de convertir les variables en longueurs.
La longueur à reporter sur un des axes pour une valeur donnée se calcule par une règle de trois :

longueur axe **x** : valeur **x** = 5 : 2 (5 mm = 2 km)

longueur axe **y** : valeur **y** = 1 : 20 (1 cm = 20 km/h)

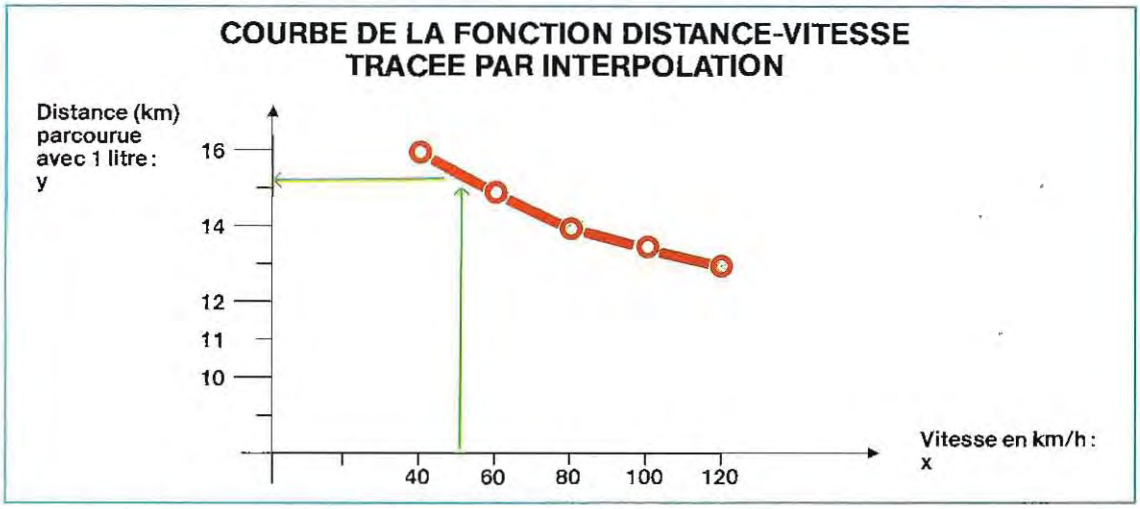
$$\text{longueur axe } \mathbf{x} = \frac{5 \times \text{valeur } \mathbf{x}}{2} = \frac{5}{2} \times \text{valeur } \mathbf{x}$$

$$\text{longueur axe } \mathbf{y} = \frac{1 \times \text{valeur } \mathbf{y}}{20} = \frac{1}{20} \times \text{valeur } \mathbf{y}$$

Ces facteurs $\frac{5}{2}$ et $\frac{1}{20}$, sont appelés **coefficients d'échelle.**
La formule générale est :

$$\text{longueur sur l'axe} = \text{coefficient d'échelle} \times \text{valeur}$$

Nous avons ainsi construit une table et un graphique grâce à la mesure d'un phénomène physique : la variation du parcours en fonction de la variation de vitesse. Et ce, à partir du choix arbitraire d'un certain nombre de vitesses (par ex., 40, 60, 80 km/h) et la mesure des kilométrages correspondants.
Nous voici donc en possession d'une série de résultats ponctuels mais nous n'avons pas déterminé la relation continue associant ces deux variables.
Cependant, ayant pris des mesures assez rapprochées, nous sommes en droit de penser qu'à une vitesse intermédiaire entre deux vitesses étudiées correspond un kilométrage compris entre les deux distances correspondantes.
Nous allons donc relier les points de coordonnées connues par des segments de droite et admettre que tous les couples (**x**, **y**) que nous aurions pu trouver expérimentalement sont situés sur la courbe ainsi obtenue.
Le graphique de cette fonction est alors terminé, et le schéma de la page 437 montre comment trouver la distance couverte à une autre vitesse (50 km/h dans cet exemple).
Cependant, la courbe tracée n'est qu'approximative, car la relation exacte entre **x** et **y** est une formule mathématique qui donne **y** en fonction de **x**.
Néanmoins, il est certain que la courbe réelle passera par les points que nous avons trouvés expérimentalement.
La définition d'une fonction reliant deux variables **x** et **y** permet de calculer la valeur de **y** associée à un **x** donné.
La variable **x**, dont on peut fixer arbitrairement la valeur (ici la vitesse), est dite indépendante.
L'autre variable (ici la distance) est au contraire dépendante de **x** puisque déduite de la formule.

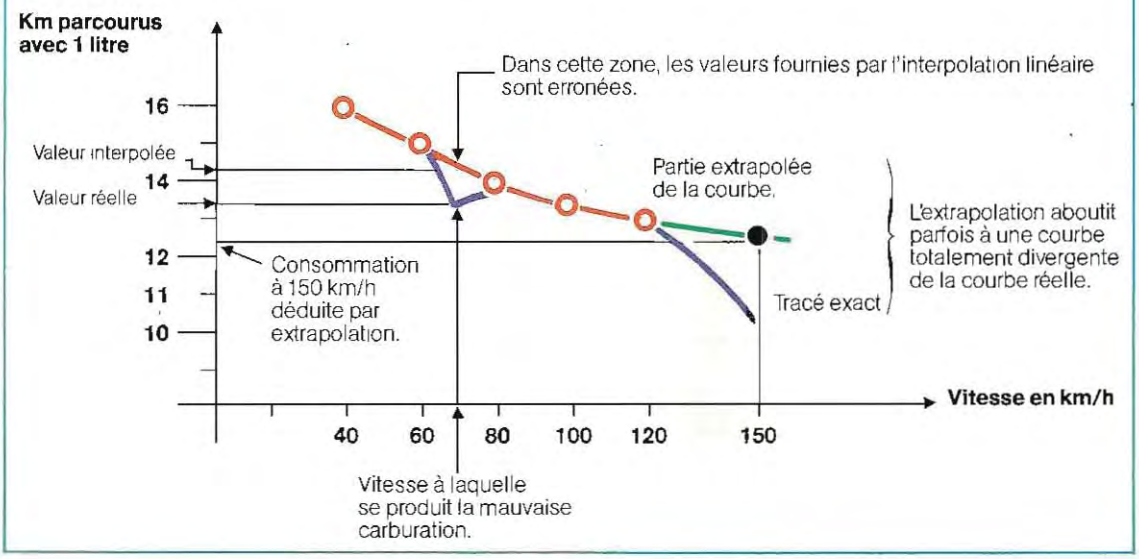


On représente généralement la **variable indépendante** sur l'axe des abscisses (**x**) et la **variable dépendante** sur l'axe des ordonnées (**y**).
Une fonction faisant dépendre une variable **y** d'une autre variable **x** se note généralement : **y = f(x)**, soit, en clair :
« les valeurs prises par la variable dépendante **y** sont fonction des valeurs de la variable indépendante **x** ».
Nous avons vu que, pour représenter graphiquement la fonction **y = f(x)** qui, dans notre cas, correspond à :
distance = f(vitesse),
on choisit des valeurs de la variable indépendante (**x**, vitesse) et on mesure les valeurs correspondantes de la variable dépendante (**y**, distance).
La liberté de choix des valeurs de **x** est toutefois limitée car il n'est pas toujours possible d'effectuer une mesure à 200 km/h (dans le cas où la voiture concernée ne peut atteindre cette vitesse).
On veillera donc à ne pas choisir des valeurs de **x** totalement arbitraires.
L'intervalle dans lequel la variable **x** peut prendre ses valeurs s'appelle le **domaine de définition de la fonction.**

Interpolation et extrapolation

Pour l'exemple que nous avons choisi, nous ne disposions que d'un nombre limité de mesures (40, 60, 80, 100 et 120 km/h, soit cinq valeurs au total) et nous avons tout de même tracé une courbe continue en reliant les points connus.
Cette opération, qui semble aller de soi, relève en réalité d'un procédé mathématique complexe : **l'interpolation linéaire.**
Interpoler signifie déduire de deux ou plusieurs résultats des valeurs intermédiaires.
C'est ce que nous avons fait sur le schéma de cette page en traçant les segments de droite qui sont censés relier tous les points correspondants aux mesures non effectuées.
En reliant les points correspondant aux vitesses de 40 et 60 km/h, nous avons effectué une interpolation linéaire entre ces deux relevés nous permettant d'estimer les consommations intermédiaires (par exemple à 50 km/h).
L'interpolation linéaire est presque toujours acceptable d'un point de vue mathématique, mais elle ne fournit des valeurs proches de la réalité que dans le cas de courbes au tracé régulier.
Imaginons, en effet, que la voiture testée présente un défaut de carburation entre 60 et 80 km/h (voir page 438).
Le pic de consommation ainsi provoqué affecte la régularité de la courbe et on commettrait une erreur grossière en utilisant l'interpolation linéaire pour déduire la distance parcourue à 70 km/h.
On peut toutefois diminuer ce risque en effectuant des mesures intermédiaires de façon à interpoler des points plus rapprochés.
En dépit des risques d'erreurs encourus, l'interpolation est très souvent utilisée dans les calculs scientifiques.
Il existe d'ailleurs d'autres types d'interpolation que l'interpolation linéaire.

INTERPOLATION ET EXTRAPOLATION



Par exemple, on approche beaucoup plus de la réalité en reliant les points par des courbes plutôt que par des droites.

Un autre procédé très employé en mathématiques est **l'extrapolation.**

La vitesse maximale que nous avons testée était de 120 km/h.

Or, si notre véhicule peut atteindre 150 km/h, le domaine de définition de la fonction distance/vitesse s'étend jusque-là aussi.

Mais puisque nous n'avons pas testé la vitesse maximale, il nous est impossible de trouver, par interpolation, les valeurs de la consommation entre 120 et 150 km/h.

Nous allons donc être obligés de le faire par extrapolation, en prolongeant la courbe intuitivement.

Cette méthode peut également s'appuyer sur des calculs mathématiques, mais si, pour interpoler, il suffit d'imaginer le tracé entre deux points connus (avec peu de chances de s'écarter beaucoup du tracé réel), l'extrapolation est beaucoup plus hasardeuse.

En effet, nous ignorons tout de la courbe après le dernier point.

L'évolution de la fonction après la dernière mesure peut très bien présenter une modification, voire une inversion, de la tendance précédente.

Justement, dans notre exemple, la consommation augmente rapidement au voisinage de la vitesse maximale, et le tracé de la

courbe chute de façon imprévisible. Dans ce cas, une extrapolation mènerait à des déductions entachées d'une erreur importante (voir le schéma de la page 438).

Représentation analytique des fonctions

La représentation graphique des fonctions permet une visualisation globale de leur tracé mais ne peut en aucun cas constituer une méthode de calcul.
On pourrait éventuellement estimer les frais de carburant à différentes vitesses, en utilisant le graphique pour déduire la consommation.
Mais ce travail ne peut être effectué par un ordinateur. Ces machines travaillent sur des données précises et sont théoriquement inaptes à « évaluer » une valeur à partir d'un graphique.
La solution est de trouver une formule mathématique dont le tracé se rapproche du graphique. L'application de cette formule permettra alors de calculer la distance correspondant à une vitesse donnée.
Ainsi, pour ce qui est de notre exemple, la formule suivante donnerait une bonne approximation de la fonction :

$$\text{km parcourus avec 1 litre} = 18.6 - 0.07 \times \text{vitesse en km/h}$$

Il suffit dès lors de reporter la vitesse dans cette formule et d'effectuer les calculs indiqués pour trouver la distance correspondante.
Cette expression arithmétique est la **représentation analytique** de la fonction.
En programmation, on doit exprimer sous forme analytique toutes les fonctions qu'on veut utiliser.
Deux cas peuvent se présenter :
soit il s'agit d'une fonction déterminée et il suffit de la transcrire en langage symbolique, soit on ne connaît que quelques points, et il faudra trouver une fonction approchante (comme nous l'avons fait avec la courbe de consommation).
Il existe alors des méthodes plus ou moins complexes selon la précision requise.
L'interpolation linéaire, ou encore régression linéaire, qui suppose une liaison rectiligne entre les points connus, est la plus simple de ces techniques.
Toute corrélation entre deux (ou plusieurs) variables peut être exprimée sous la forme d'une fonction.
Leur symbolisation est **y = f(x)**, mais on pourrait très bien remplacer x et y par d'autres caractères.
Tout langage de programmation possède des fonctions intrinsèques et un symbolisme propre, mais on retrouve la plupart du temps cette structure :

$$\text{variable indépendante} = \text{NOM} \quad \text{variable dépendante}$$

où NOM est le « mnémonique » correspondant à la fonction (sous la forme d'un sigle ou d'une abréviation).

Aujourd'hui, les possibilités graphiques des micro-ordinateurs personnels permettent une représentation claire de fonctions complexes. Ici : un histogramme en trois dimensions.

Olivetti

Les fonctions du Basic

Le Basic est un des langages les plus riches en fonctions prédéfinies. Destinées pour une bonne part au traitement des chaînes de caractères, ces fonctions facilitent considérablement le travail du programmeur qui les maîtrise bien. Nous vous conseillons donc une lecture attentive de ce chapitre.
Les fonctions du Basic peuvent être classées en trois groupes principaux :

- les fonctions mathématiques ;
- les fonctions permettant le traitement des chaînes de caractères ;
- les fonctions spéciales (mémoire et E/S).

L'application de certaines fonctions mathématiques nécessite des notions de trigonométrie et de calcul logarithmique. Ceci concerne surtout les applications scientifiques et nous ne présenterons ici ces instructions qu'à titre d'information. Les lecteurs intéressés par ce sujet pourrons utilement se référer à un manuel d'analyse et de trigonométrie.

Les fonctions mathématiques

Ces fonctions renvoient un résultat numérique (Y) après traitement de l'argument proposé (X). Elles seront présentées par ordre de difficulté croissante afin de faciliter la lecture de ce chapitre.
Les premières fonctions permettent de modifier la présentation d'un nombre (arrondi, extraction de la partie entière, etc.). Viennent ensuite les fonctions mathématiques courantes suivies des transformations trigonométriques et exponentielles. L'argument à communiquer est noté N ou R selon qu'il s'agit d'un entier ou d'un réel. Cette distinction est purement formelle car le Basic effectue une conversion automatique en cas d'erreur.
Enfin, contrairement au Basic compilé, le Basic interprété n'accepte généralement pas les arguments en double précision.

ABS(R). Cette fonction renvoie la valeur absolue (c'est-à-dire positive) de R. L'argument peut être une expression arithmétique. Ainsi :

```
Y = ABS (-10)
Z = ABS(3*(7-5)-2)
```

sont deux écritures correctes de cette instruction. La première affecte à Y la valeur 10 et dans la seconde, on donne à Z la valeur 4. La fonction ABS(R) s'avère utile avant l'application d'instructions dont l'argument doit être positif.
On pourrait, bien sûr, arriver au même résultat en testant le signe du paramètre et en le multipliant par –1 s'il est négatif. Le nombre obtenu serait alors toujours positif. Il est cependant plus rapide d'utiliser l'instruction ABS(R).
On peut aussi déterminer si une variable X est comprise dans l'intervalle [–0.01, 0.01] par cette instruction :

```
XI = ABS(X)  'XI est la valeur absolue de X
IF  XI < = 0.01 THEN ...
```

ou, plus directement :

```
IF ABS(X) < = 0.01 THEN ...
```

CDBL(R). Convertit R en un nombre en double précision. Exemple :

```
10  R = 56.8
20  DR = CDBL(R)
```

La valeur de R est affectée à une variable DR de format double précision. Ce format réserve pour la variable deux fois plus de place en mémoire, et autorise ainsi l'utilisation de beaucoup plus de chiffres significatifs.
On emploie cette instruction avant les calculs comportant à la fois des variables en double et en simple précision. On a alors un format homogène, ce qui limite les risques d'erreur.

CINT(R). Transforme un réel en entier avec arrondissement de sa partie décimale.
Le résultat étant un entier doit obligatoirement être compris entre –32768 et +32767, sous peine d'un dépassement de capacité (« overflow error »).
Ainsi :

```
A = CINT(7.51) et
B = CINT(3574.2)
```

sont correctement formulés alors que :

```
C = CINT(82754.3)
```

provoquera une interruption du programme.

L'instruction CINT est donc utilisable dans l'intervalle des nombres entiers. Au-delà, on a recours à des méthodes plus complexes que nous étudierons plus loin.

CSNG(R). C'est la fonction inverse de CDBL(R) ; elle produit le passage de double en simple précision. Exemple :

```
R = 135.7943 'constante en double précision
M = CSNG(R) 'affecte à M la valeur de R en simple précision
```

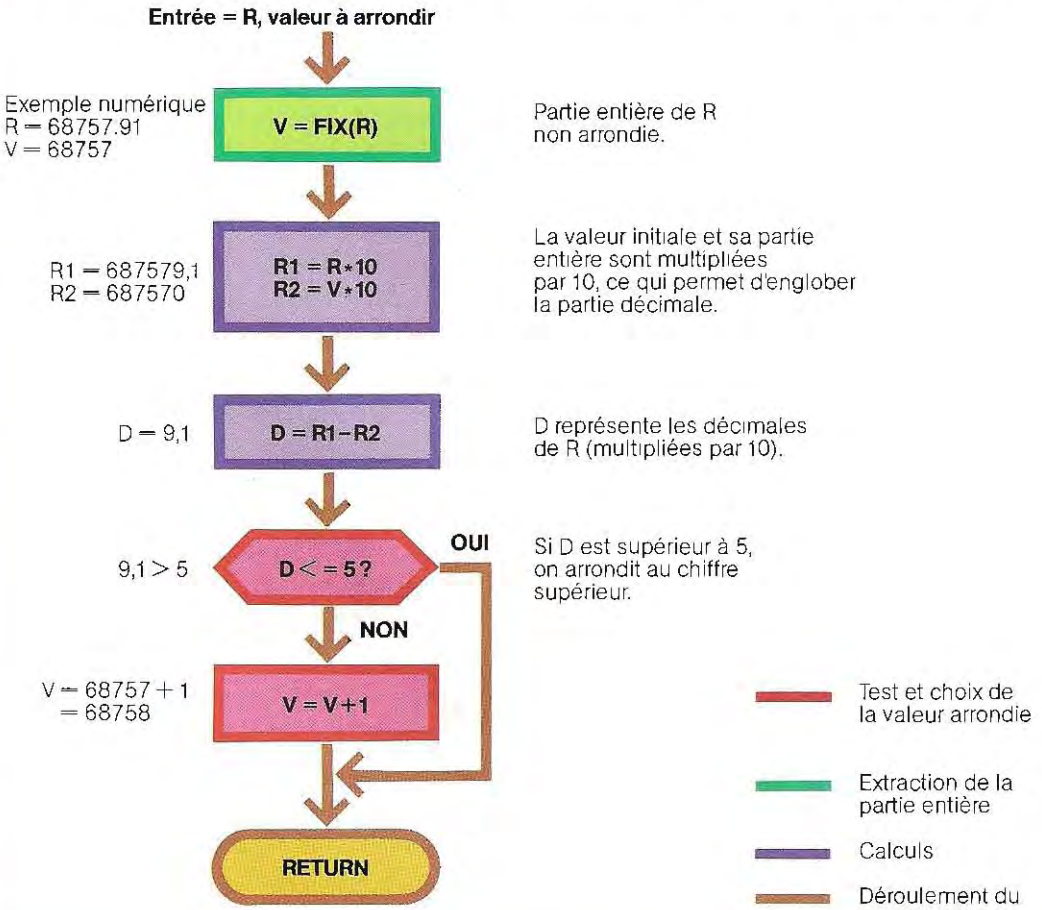
On utilise la fonction CSNG(R) pour alléger l'espace mémoire, ou pour rendre homogènes les facteurs d'un calcul.

FIX(R). Tronque la valeur de R à sa partie entière. Ainsi :

M = FIX(572.85) donne M = 572
M = FIX(–374.6) donne M = –374

Quoique sans décimales, la valeur rénvoyée est un réel. On peut donc utiliser cette fonction en dehors de l'intervalle des entiers (–32768, +32768), et notamment créer un équivalent de la fonction CINT hors de cet intervalle.
Il suffit pour cela d'utiliser l'instruction FIX qui donne la partie entière du nombre, puis d'examiner la partie décimale de l'argument pour savoir si l'on arrondit, ou non, au chiffre supérieur. L'organigramme de cette routine est présenté ci-dessous et son listage page 442. Elle n'est applicable qu'aux nombres positifs,

ORGANIGRAMME DU SOUS-PROGRAMME POUR L'ARRONDI DES NOMBRES



PROGRAMME POUR L'ARRONDI DES NOMBRES

```
10 A$="$$$$$$$$$$$.$"
20 INPUT "VALEUR A ARRONDIR";R
30 '
40 GOSUB 1000
50 '
60 LPRINT "VALEUR A ARRONDIR R = ";:LPRINT USING A$;R
70 LPRINT "VALEUR ARRONDIE    V = ";:LPRINT USING A$;V
80 LPRINT
90 GOTO 20
100 END
1000 ' * SOUS-PROGRAMME POUR L'ARRONDI *
1010 V=FIX(R)     ' PARTIE ENTIERE DE R NON ARRONDIE
1020 R1=R*10
1030 R2=V*10
1040 '
1050 D=R1-R2      ' D CONTIENT LES DECIMALES DE R
1060 IF D<=5 THEN RETURN ELSE V=V+1 : RETURN
```

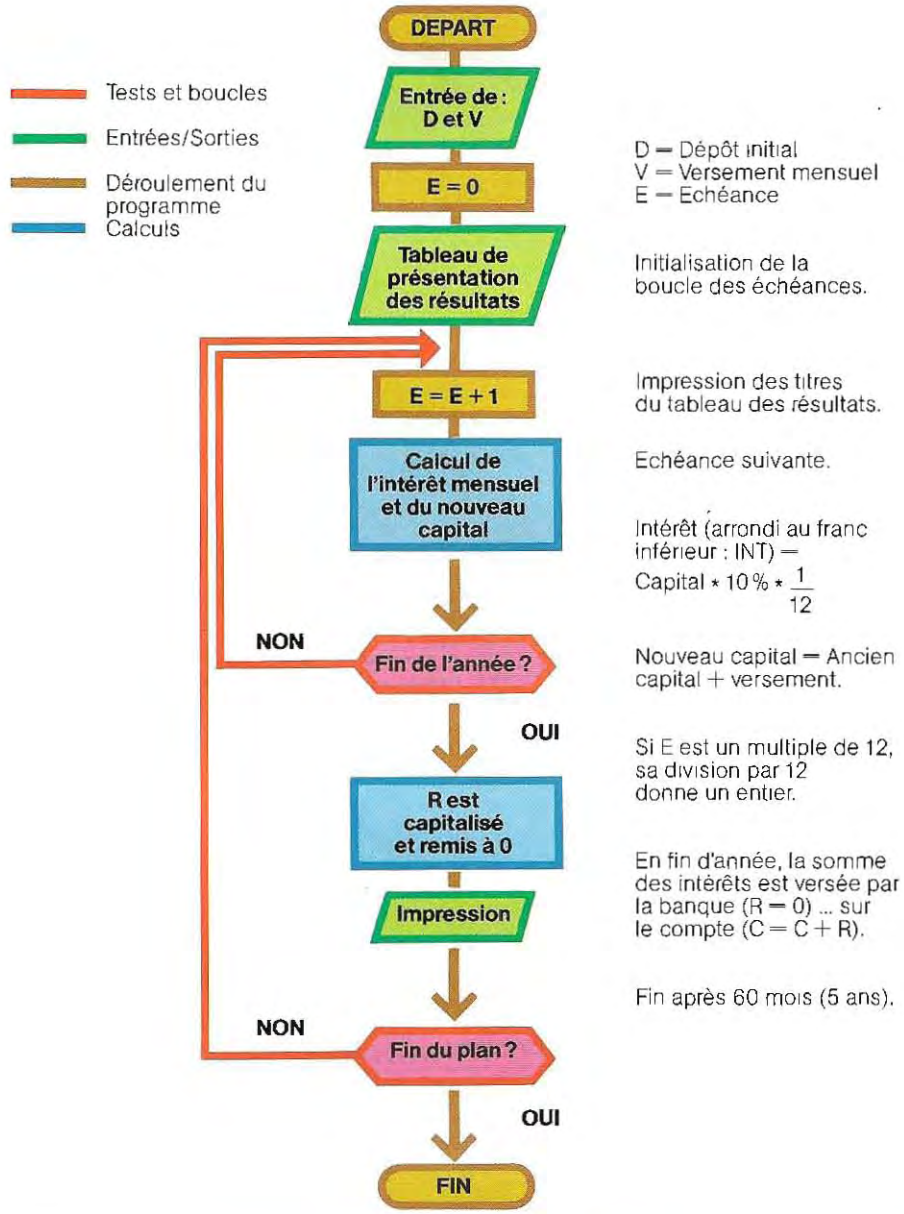
```
VALEUR A ARRONDIR R =          123.5
VALEUR ARRONDIE   V =          123.0

VALEUR A ARRONDIR R =          123.6
VALEUR ARRONDIE   V =          124.0
```

PROGRAMME DE SIMULATION D'UN PLAN D'EPARGNE-LOGEMENT

```
70 A$="£££££££££"    ' MASQUE D'IMPRESSION (CETTE INSTRUCTION
80 '                  SERA EXPLIQUEE PAR LA SUITE)
90 INPUT "DEPOT INITIAL:";D
100 C=D    ' CAPITAL EGAL AU DEPOT
110 INPUT "VERSEMENTS MENSUELS:";V
120 E=0    ' INITIALISATION DE LA BOUCLE : ECHEANCE ZERO
140 ' PRESENTATION DU TABLEAU SUR L'IMPRIMANTE
150 LPRINT "EVOLUTION ANNUELLE DU CAPITAL SUR UN PLAN D'EPARGNE-LOGEMENT"
160 LPRINT "    (TAUX : 10% SUR 5 ANS)
170 LPRINT
180 LPRINT "AU TERME DE:", "CAPITAL", "SOMMES VERSEES"
200 LPRINT
210 E=E+1          ' COMPTEUR DES ECHEANCES
230 I=INT(C*0.1/12)     ' INTERET MENSUEL
240 R=R+I          ' LA REMUNERATION AUGMENTE
260 C=C+V          ' VERSEMENT MENSUEL
270 IF INT(E/12)<>E/12 GOTO 210
280 C=C+R      ' REMUNERATION CAPITALISEE TOUS LES ANS
290 R=0
300 ' IMPRESSION DES RESULTATS
310 ' *** L'INSTRUCTION "LPRINT USING " SERA EXPLIQUEE ULTERIEUREMENT ***
320 LPRINT "ANNEE ";E/12,:LPRINT USING A$;C,:LPRINT USING A$;D+E*V
330 IF E<60 GOTO 210
360 END
```

```
EVOLUTION ANNUELLE DU CAPITAL SUR UN PLAN D'EPARGNE-LOGEMENT
    (TAUX : 10% SUR 5 ANS)

AU TERME DE :        CAPITAL        SOMMES VERSEES

ANNEE : 1             23540             22000
ANNEE : 2             30444             34000
ANNEE : 3             54834             46000
ANNEE : 4             72860             58000
ANNEE : 5             92690             70000
```

PROGRAMME DE SIMULATION D'UN PLAN EPARGNE-LOGEMENT



mais il est très simple de l'étendre aux nombres négatifs. Rappelons en effet que la valeur arrondie de −374.6 n'est pas −375 mais −374 et qu'il convient dans ce cas d'inverser la logique du programme.

INT(R). Renvoie l'entier immédiatement inférieur à R. Ainsi :

N = INT (76.93) donne N = 76

N = INT (−21.2) donne N = −22 (et non −21)
N = INT (−21.0) donne N = −21

A titre d'illustration, nous vous proposons un programme simulant un plan d'épargne-logement. L'évolution du capital sera calculée annuellement en fonction du dépôt initial et des versements mensuels.
L'instruction INT est utilisée lors du calcul des

intérêts (arrondissement au franc inférieur) et pour déterminer la fin d'une année.
Chaque mois sont évalués les intérêts du capital précédent, puis le versement mensuel vient s'ajouter à celui-ci.
Au bout d'un an, les intérêts sont capitalisés et l'ordinateur indique la nouvelle valeur du placement.
En réalité, les intérêts sont intégrés au capital le 31 décembre et non un an après ouverture du plan.
Cette simplification n'induit toutefois qu'une erreur minime : une centaine de francs en plus (pour les valeurs de notre exemple) si le plan a été (par ex.) ouvert en juin.
Vous trouverez l'organigramme et le listage de cette application aux pages 443 et 442.

SNG(R). Renseigne sur le signe de R par un code égal à 1 si R est positif, à 0 si R est nul et à −1 si R est négatif. Ainsi :

Y = SNG(28) donne Y = 1
Y = SNG(0) donne Y = 0
Y = SNG(−75) donne Y = −1

SQR(R). Renvoie la racine carrée de R. Ainsi :

Y = SQR(16) donne Y = 4
Y = SQR(15) donne Y = 3.87298

Le nombre R doit être supérieur ou égal à zéro car les nombres négatifs n'ont pas de racine carrée dans l'ensemble des réels.

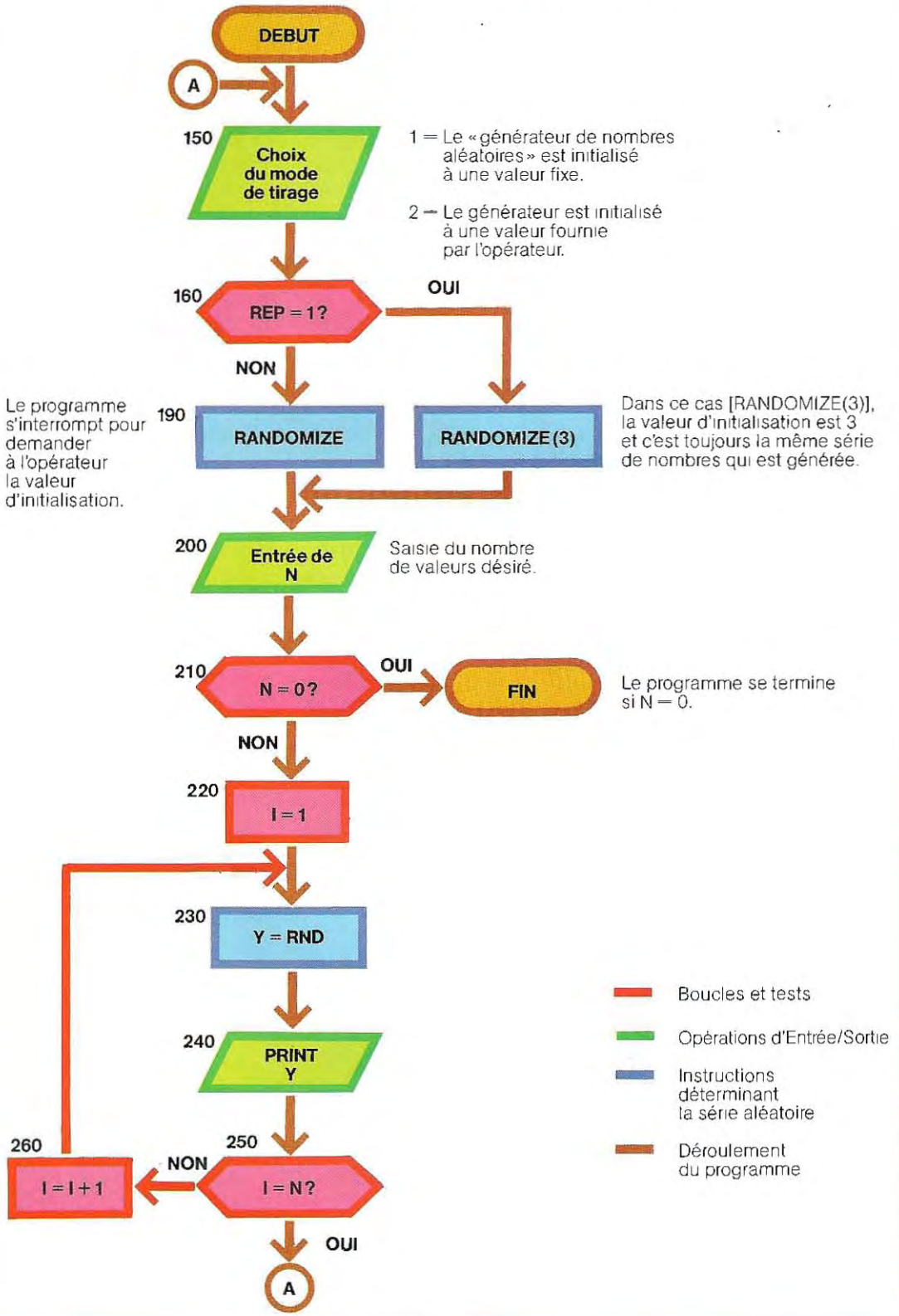
RND(R). Cette fonction génère un nombre aléatoire (random) compris entre 0 et 1.
La notion de nombre aléatoire est assez complexe, aussi renvoyons-nous aux manuels de statistiques le lecteur qui souhaite en étudier une définition plus détaillée.
Du point de vue de l'utilisateur, la fonction RND(R) active un algorithme dont l'application répétée engendre une série de nombres tirés « au hasard ». La probabilité d'obtenir deux fois le même chiffre est donc très faible.
Le tirage de nombres aléatoires se déroule en deux étapes. Un générateur de nombres aléatoires est d'abord initialisé par l'instruction RANDOMIZE(N) (avec N entier).
Cette valeur N sera ensuite utilisée comme

PROGRAMME DE GENERATION DE NOMBRES ALEATOIRES

```
50 BI$=CHR$(27)+"*"+CHR$(7)   'NETTOYAGE D'ECRAN ET EMISSION D'UN SON
60 PRINT BI$
70 PRINT "*** PROGRAMME DE GENERATION DE NOMBRES ALEATOIRES ***"
80 PRINT
90 PRINT "       1) = GENERATEUR DE NOMBRES ALEATOIRES EST "
100 PRINT "          INITIALISE AUTOMATIQUEMENT "
110 PRINT
120 PRINT "       2) = GENERATEUR DE NOMBRES ALEATOIRES EST "
130 PRINT "          INITIALISE MANUELLEMENT "
140 PRINT
150 INPUT "TAPEZ VOTRE SELECTION";REP
160 IF REP=1 THEN RANDOMIZE(3):GOTO 200
180 PRINT BI$
190 RANDOMIZE     'VALEUR D'INITIALISATION DEMANDEE A L'OPERATEUR
200 INPUT "COMBIEN DE NOMBRES ALEATOIRES DESIREZ-VOUS?";N
210 IF N=0 THEN STOP
215 LPRINT "PROGRAMME DE GENERATION DE NOMBRES ALEATOIRES"
220 I=1
230 Y=RND
240 LPRINT Y
250 IF I=N THEN LPRINT:GOTO 80
260 I=I+1
270 GOTO 230
280 END
```

```
PROGRAMME DE GENERATION DE NOMBRES ALEATOIRES

 .88588
 .484668
 .508528
```

GENERATION DE NOMBRES ALEATOIRES



argument pour la génération d'une série de nombres à chaque appel de RND(R).
Si l'on omet RANDOMIZE, l'initialisation donnera toujours le même résultat et les séries successives seront identiques.
La seconde étape est le processus de génération proprement dit lancé par RND.
Cette instruction s'emploie telle quelle ou avec un argument sous la forme RND(R). Tout argument non nul induira le tirage d'une nouvelle série ; par contre la série précédente sera répétée si R = 0. L'instruction RANDOMIZE peut aussi être employée sans argument ; l'ordinateur interrompra simplement l'exécution pour le demander à l'écran.
Le listing de la page 444 reproduit un programme de génération de nombres aléa-

Test 13

1 / Une ligne de programme comporte le calcul suivant : A=B/(C−D) ; cette instruction peut être à l'origine d'une interruption de programme. Pourquoi ?

2 / Comment peut-on prévenir cette erreur ?

3 / Certaines de ces instructions sont incorrectes. Lesquelles ?
a) IF K = 2.5 THEN 300
b) IF L+M = 6 GOTO 150
c) IF L = 5 RETURN
d) A=−5:B=2:K=A−B:ON K GOTO 10,20,30

4 / Imaginer un programme qui recherche si la variable V est comprise entre 2.6 et 15.8 (dans les deux cas : bornes inclues et bornes exclues).

5 / Quelles valeurs sont affectées aux variables E et F après l'exécution du programme suivant ?

```
10 A=-18.75
20 B=20
30 C=FIX(A)
40 D=INT(A)
50 E=B+C
60 F=B+D
```

6 / Imaginer un programme qui contrôle le signe de la variable V, puis qui présente le résultat sous la forme d'une phrase du type : supérieure à 0, inférieure à 0, égale à 0.

7 / Quelle instruction doit-on ajouter à ce programme pour éviter une erreur ?

```
10 DEFINT A-Z
20 B=-9
30 C=SQR(B)
```

Les solutions du test se trouvent page 451.

toires, correspondant à l'organigramme schématisé page 445.

ATN(Y). L'argument de cette fonction, quoique réel, est symbolisé par Y pour éviter la confusion avec la fonction TAN présentée plus loin.
ATN(Y) fournit l'angle trigonométrique dont la tangente est Y (fonction arctangente).
La valeur renvoyée est donc exprimée en radians et comprise entre $-\pi/2$ et $+\pi/2$.
Le symbole π désigne le rapport entre la circonférence d'un cercle et son diamètre : π = circonférence/diamètre = 3.1415926...
Y peut être donné en double précision mais le résultat est presque toujours en simple précision.

Les fonctions trigonométriques

COS(R). C'est la fonction cosinus. Elle renvoie le cosinus de R qui doit être exprimé en radians. Le résultat est également en simple précision.
L'unité angulaire généralement employée est le degré. On peut toutefois convertir les degrés en radians grâce à cette formule :

$$\text{Angle en radians} = \frac{\pi \times \text{angle en degrés}}{180} = 0.01745 \times \text{angle en degrés}$$

SIN(R). C'est la fonction sinus. Elle calcule le sinus de R qui doit là aussi être exprimé en radians.

TAN(R). C'est la fonction tangente. Elle calcule la tangente de R, toujours exprimé en radians. Notons que les fonctions TAN(R) et ATN(R) sont réciproques l'une de l'autre : TAN(R) donne la tangente de l'angle R, et ATN(Y) donne l'angle dont Y est la tangente. Ou, de façon plus concise :

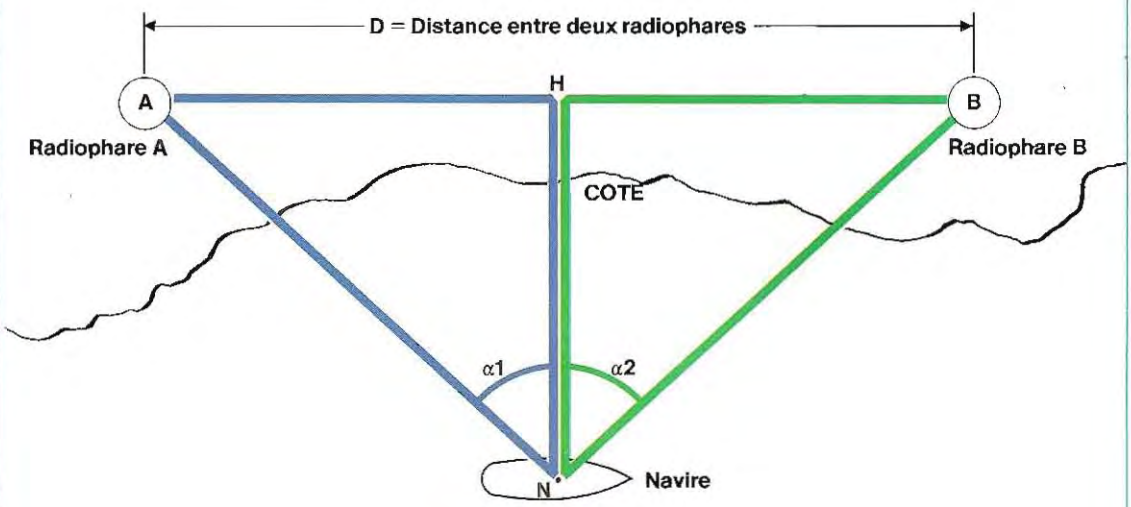
Y = TAN(R)
Y est la valeur de la tangente de l'angle R.
R = ATN(Y)
R est la mesure de l'angle qui a Y pour tangente.

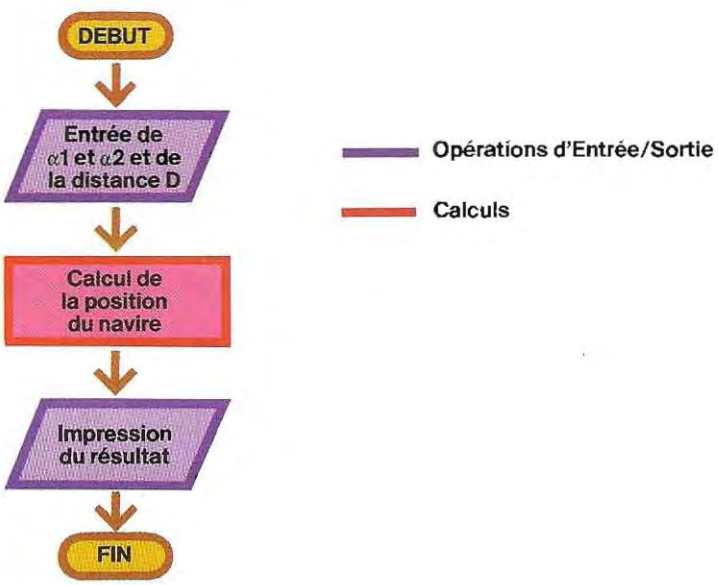
Nous vous présentons ci-dessous le listage et l'exécution d'un programme de démonstration des fonctions trigonométriques.

EXEMPLE D'UTILISATION DES FONCTIONS TRIGONOMETRIQUES

```
40  ' RD = ANGLE EXPRIME EN DEGRES
50  '
60  ' RR = MEME ANGLE CONVERTI EN RADIANS
70  '
100 INPUT "ANGLE (EN DEGRES)";RD
110 IF RD=90 GOTO 1000    ' COS(90°)=0 ; ERREUR EN LIGNE 225
120 IF RD>180 GOTO 1000
130 '
140 ' * CONVERSION EN RADIANS *
150 '
160 RR=.01745*RD
170 '
180 '
190 ' * FONCTIONS TRIGONOMETRIQUES *
200 '
210 S=SIN(RR)          ' SINUS
220 C=COS(RR)          ' COSINUS
230 T=S/C              ' TANGENTE = SINUS/COSINUS
240 A=ATN(T)           ' ARCTANGENTE; DOIT CORRESPONDRE A RR
250 '
260 ' * IMPRESSION *
270 '
280 LPRINT "ANGLE EN DEGRES =";RD,"EQUIVALENT EN RADIANS =";RR
290 LPRINT "SINUS =";S,"COSINUS =";C,"ARCTANGENTE =";A
295 LPRINT
300 GOTO 100
310 '
1000 END

ANGLE EN DEGRES = 20          EQUIVALENT EN RADIANS = .349
SINUS = .341958       COSINUS = .939715       ARCTANGENTE = .349
```

APPLICATION DES FONCTIONS TRIGONOMETRIQUES A UN PROBLEME DE TRIANGULATION





Applications des fonctions trigonométriques. Les fonctions trigonométriques sont indispensables pour résoudre les problèmes de triangulation ; et notamment dans toute évaluation de distance, d'altitude, etc., à partir de simples mesures d'angle. C'est, par exemple, la méthode employée pour déterminer la position d'un navire à l'aide des signaux émis par deux balises radio.

Dans la réalité, les positions des radiophares et du navire sont caractérisées par leur latitude et leur longitude dans le système géodésique des méridiens et des parallèles. Le calcul se trouve ainsi simplifié. Notre exemple n'ayant qu'une valeur d'illustration, nous négligerons ces possibilités et nous nous contenterons de calculer la distance du bateau à la côte.

Le radiogoniomètre, récepteur spécial installé à bord du bateau, permet de déterminer les angles des signaux émis par les radiophares ($\alpha 1$ et $\alpha 2$ sur le schéma ci-dessus).

On connaît ces deux angles et la distance D séparant les radiophares. Il est alors possible de trouver la distance du bateau à la côte (NH) par des calculs simples sur les triangles rectangles NHA (en bleu) et NHB (en vert).
Voici la formule à employer :

$$NH = \text{distance de la côte} = \frac{\text{distance entre les radiophares}}{tg(\alpha_1) + tg(\alpha_2)}$$

Les expressions $tg(\alpha_1)$ et $tg(\alpha_2)$ désignent les tangentes des angles α_1 et α_2.

Le programme devra donc exécuter les opérations suivantes :

1 / entrée des angles α_1 et α_2 et de la distance D ;
2 / conversion des angles en radians ;
3 / calcul de la tangente de chaque angle ;
4 / calcul de la distance du bateau à la côte ;
5 / présentation du résultat.

Le listage de ce programme se trouve ci-dessous.

Dans les cas où la fonction tangente n'est pas disponible, il est très facile de la déduire de cette formule :

$$tg(R) = \frac{SIN(R)}{COS(R)}$$

Cet exemple montre comment contourner les carences de certains BASIC simplifiés. Quand ne sont disponibles que les trois fonctions sinus, cosinus et cotangente, c'est en effet au programmeur de définir lui-même les autres. Nous reviendrons sur ce point en fin de chapitre.

EXP(R). Calcule la valeur exponentielle de R. En d'autres termes, l'instruction :

Y = EXP(R)
affecte à Y la valeur de e^R. Ainsi, pour R = 2, on a :

Y = EXP(2), $Y = e^2 = 2.7183^2 = 7.389...$

Rappelons que le nombre e de valeur 2.7183 est la base des logarithmes népériens.

CALCUL DE LA DISTANCE D'UN NAVIRE A LA COTE

```
30 PRINT "N.B. On doit entrer les angles en degrés"
40 '
50 INPUT "ALPHA 1";I
55 A=I  ' MEMORISATION DANS "A" DE LA VALEUR DE ALPHA 1 EN DEGRES
60 INPUT "ALPHA 2";II
65 B=II ' MEMORISATION DANS "B" DE LA VALEUR DE ALPHA 2 EN DEGRES
70 INPUT "DISTANCE ENTRE LES RADIOPHARES";D
80 ' CONVERSION DES DEGRES EN RADIANS
90 I=I*.01745
100 II=II*.01745
110 '
120 ' CALCUL DE LA DISTANCE NH=D/(TAN(I)+TAN(II))
130 NH=D/(TAN(I)+TAN(II))
140 '
150 '
160 ' * INSTRUCTIONS D'IMPRESSION *
170 LPRINT "DISTANCE ENTRE LES RADIOPHARES     ";D
180 LPRINT "ANGLE ALPHA 1 = ";A
190 LPRINT "ANGLE ALPHA 2 = ";B
200 LPRINT "N.B. LES ANGLES SONT EXPRIMES EN DEGRES"
210 LPRINT              ' SAUT DE LIGNE
220 LPRINT "DISTANCE DU NAVIRE A LA COTE NH = ";NH
230 LPRINT:LPRINT     ' DOUBLE SAUT DE LIGNE
240 GOTO 50
250 END
```

```
DISTANCE ENTRE LES RADIOPHARES     1455
ANGLE ALPHA 1 = 12,1555
ANGLE ALPHA 2 = 39,4546
N.B. LES ANGLES SONT EXPRIMES EN DEGRES

DISTANCE DU NAVIRE A LA COTE NH = 1401,54
```

Cette fonction est bien entendu réservée à des applications scientifiques. Elle est présentée de façon détaillée dans les manuels d'analyse mathématique.

LOG(R). Renvoie la valeur du logarithme naturel (ou népérien) de R.
Nous ne nous attarderons pas sur la signification du terme logarithme. Rappelons simplement que R doit être strictement positif.

Fonctions permettant le traitement des chaînes de caractères

Une particularité du Basic est sa vaste gamme de commandes concernant les chaînes de caractères. L'emploi de ces instructions procure une grande souplesse dans la présentation des tableaux et facilite la rédaction des traitements de fichiers. Le nombre et la syntaxe de ces commandes varient selon les différentes versions de Basic. C'est pourquoi nous nous limiterons, dans ce chapitre, à étudier les fonctions les plus courantes du Basic 80.
Nous aborderons par la suite certaines structures ou fonctions, absentes de cette version standard, et qui permettent, entre autres, la manipulation de morceaux de chaîne. Rappelons pour finir que certains Basic n'autorisent le transfert sur disque que pour des données en ASCII. On doit alors convertir en caractères toutes les valeurs numériques avant de les sauvegarder et effectuer la conversion inverse (d'ASCII en numérique) après la relecture. Les méthodes de conversion sont développées à la fin de ce chapitre.

ASC(A$). Fournit la valeur numérique (selon le code ASCII) correspondant au premier caractère de la chaîne A$. Ainsi :

```
10  A$ = "17ABC"
(la chaîne A$ contient les caractères 1, 7, A, B et C)
20  X = ASC(A$)
(le code du premier caractère est affecté à X)
30  PRINT X
      49
(la valeur de X s'affiche)
```

Ici, la fonction X = ASC(A$) a affecté à X la valeur 49 qui est le code ASCII du caractère 1 (les symboles numériques contenus dans une chaîne sont traités comme des caractères). On trouvera au bas de cette page le listing du programme correspondant.

CHR$(N). Crée une chaîne d'un seul caractère (celui dont N est le code en ASCII).
Ainsi, par exemple :

```
10  N = 65
20  B$ = CHR$(N)
30  PRINT B$
      A
```

65 est le code de la lettre A. Vous trouverez page 452 le listing d'un programme qui associe à une donnée N le caractère correspondant. Ce programme vérifie que N correspond à un code existant. Il imprime ensuite le caractère concerné en majuscule et en minuscule si on le lui a précisé.

EXEMPLE D'UTILISATION DE L'INSTRUCTION ASC(A$)

```
10 ' **EXEMPLES D'UTILISATION DE L'INSTRUCTION ASC(A$) **
15 '       FICHIER = INSTASC
20 DEFINT A-Z       ' TOUTES LES VARIABLES SONT DEFINIES COMME DES ENTIERS
30 INPUT "ENTREZ UNE CHAINE QUELCONQUE";A$
40 IF A$ = "FIN" GOTO 100     ' ON ARRETE LE PROGRAMME EN ENTRANT LE MOT FIN
50 N = ASC(A$)       ' EXTRACTION DU CODE DU PREMIER CARACTERE
60 LPRINT A$,N       ' IMPRESSION DE LA CHAINE ET DU CODE
80 LPRINT            ' SAUT DE LIGNE POUR AERER LE TEXTE
90 GOTO 30
100 END

ABCDEFGT          65

BCFGGHYK          66

CANNES            67

A                 65
```

Solutions du test 13

1-2 / L'erreur se produit quand C = D. La différence C – D devient alors nulle et la division par zéro est impossible. On peut éliminer cette erreur en « sautant » le calcul quand C = D. On peut écrire le programme de cette façon :

```
210  IF C=D GOTO 2000
220  A=B/(C-D)
225  ' * Point de rebranchement *
... suite des instructions du programme ...
2000 ' * * Erreur * *
2010 PRINT "Erreur : division par zéro"
2020 INPUT "Tapez 1 pour continuer";N
2030 IF N=1 GOTO 225
2040 STOP
```

Lorsque la condition C = D est vérifiée, le programme est détourné ligne 2000, affiche le message d'erreur (2010) et demande s'il faut continuer (2020). Si la réponse est affirmative (N = 1, 2030), il retourne exécuter les instructions 225 et suivantes. Le calcul de la ligne 220 n'est donc pas effectué et le programme ne s'interrompt pas.

---

3 / Les instructions c) et d) sont erronées.

La ligne c) est une instruction de retour conditionnel. Sa syntaxe est IF ... THEN ... THEN ne peut être omis que s'il est suivi de GOTO. L'écriture correcte est donc : IF L = 5 THEN RETURN.

A la ligne d), on veut utiliser la valeur de K pour déterminer le branchement à l'instruction 10, 20 ou 30. Or, l'instruction ON n'accepte que des paramètres positifs. De plus, cette ligne est totalement illogique puisque K est obtenu par la différence de deux constantes (A et B) et est donc elle-même une constante.

---

4 / Il faut vérifier que V est à la fois (AND) supérieur à 2.6 et inférieur à 15.8 :

```
IF 2.6<V AND V<15.8 THEN PRINT "Inclus"
```

Cette instruction exclut les bornes de l'intervalle. Pour les inclure, il suffit d'écrire :

```
IF 2.6<=V AND V<=15.8 THEN PRINT "Inclus"
```

---

5 / L'instruction C = FIX(A) supprime les décimales de A et donne donc C = –18. La ligne 40 affecte à D le premier entier inférieur à A et donc –19 (attention au signe : –18 est supérieur à –18.75).

En conséquence : E = 20 – 18 = 2 F = 20 – 19 = 1

---

6 / Il faut utiliser l'instruction SNG(V). On peut rédiger ainsi le programme :

```
10 K=SNG(V)
20 IF K=-1 THEN PRINT "Inférieur à zéro"
30 IF K=0 THEN PRINT "Egal à zéro"
40 IF K=1 THEN PRINT "Supérieur à zéro"
```

---

7 / Un nombre négatif n'a pas de racine carrée dans l'ensemble des réels. Il faut donc ajouter une instruction qui rende B positif quelle que soit sa valeur. On pourra intercaler cette ligne :

```
25 B=ABS(B)
```

PROGRAMME D'UTILISATION DE L'INSTRUCTION CHR$

```
10 ' ** PROGRAMME D'UTILISATION DE L'INSTRUCTION CHR$ **
15 '    FICHIER = INSTCHR
17 BI$=CHR$(27)+"+"+CHR$(7)
18 PRINT BI$        ' VIDE L'ECRAN ET EMET UN BIP
20 PRINT "DESIREZ-VOUS UNE TRANSCRIPTION EN MINUSCULE (OUI OU NON) ?"
25 '
30 INPUT REP$
40 IF REP$<>"OUI" AND REP$<>"NON" THEN GOTO 300
45 ' DEBUT DE LA BOUCLE
46 PRINT BI$
47 PRINT "TAPER (0) POUR ARRETER"
50 INPUT "NOMBRE A CONVERTIR EN CARACTERE";N
60 IF N=0 GOTO 1000
70 IF 47<N AND N<91 GOTO 200
80 IF 96<N AND N<123 GOTO 90
85 PRINT "ERREUR : CE CODE NE CORRESPOND A AUCUN CARACTERE":GOTO 50
90 C$ = CHR$(N)
100 LPRINT "VALEUR NUMERIQUE ";N,C$
110 GOTO 45
200 C$ = CHR$(N)
205 IF 47<N AND N<58 THEN B$="":GOTO 250
210 IF REP$="OUI" THEN GOTO 220 ELSE B$="":GOTO 230
220 B$=CHR$(N+32)
230 LPRINT "VALEUR NUMERIQUE ";N,B$,C$
240 GOTO 45
300 PRINT "ERREUR : VOUS DEVEZ REPONDRE PAR OUI OU PAR NON"
301 GOTO 25
1000 END
```

```
VALEUR NUMERIQUE 65          a          A
VALEUR NUMERIQUE 66          b          B
VALEUR NUMERIQUE 67          c          C
VALEUR NUMERIQUE 68          d          D
```

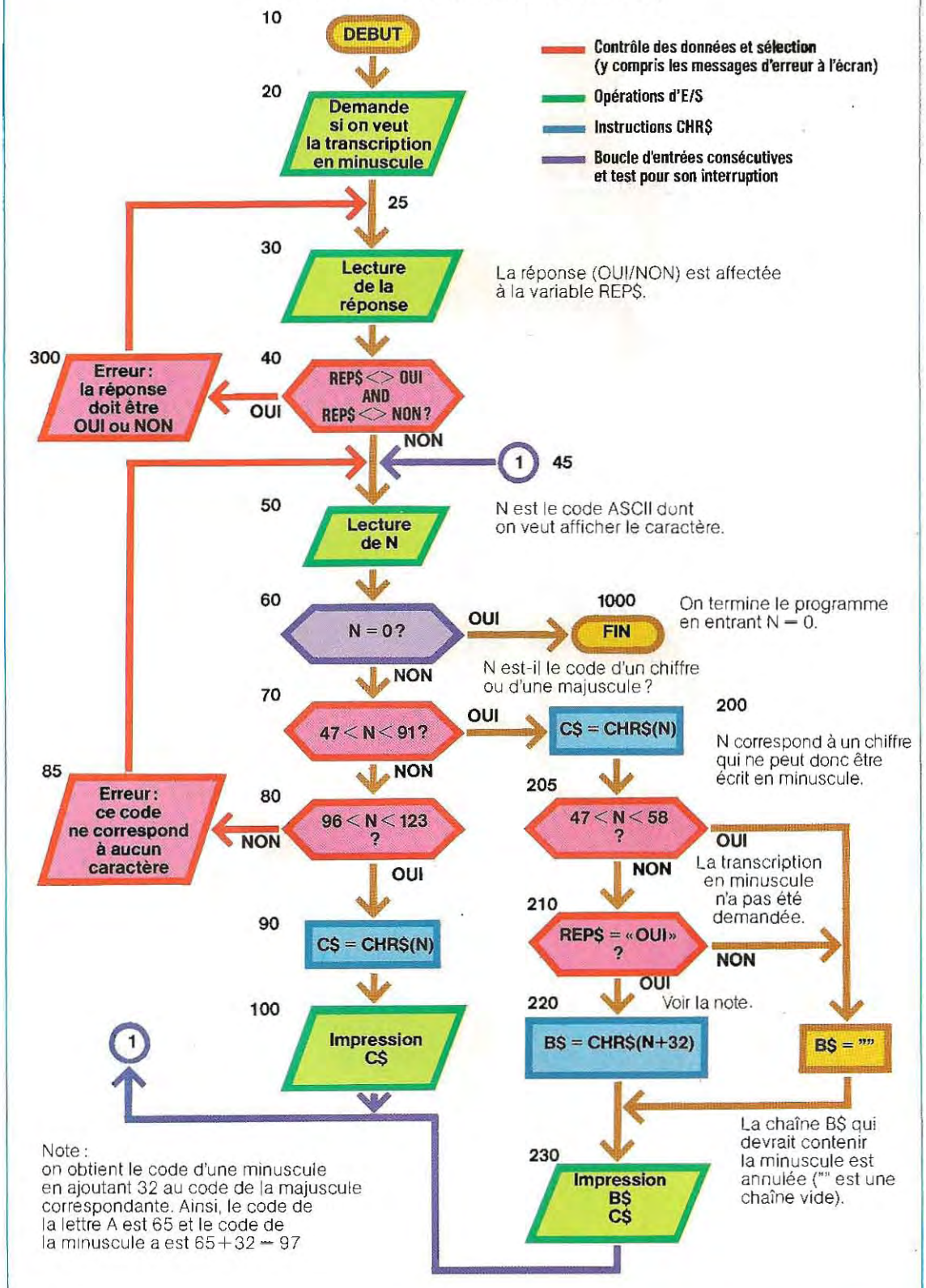
L'organigramme de ce programme se trouve page 453.

On utilise surtout l'instruction CHR$ pour transmettre des caractères de contrôle à l'écran et à l'imprimante. Ainsi, par exemple, la chaîne A$ = CHR$(12) contient un CTRL-L ou FF. La sortie de cette chaîne sur l'imprimante n'écrira rien mais produira un saut de page. Nous reviendrons plus en détail sur ce sujet par la suite.

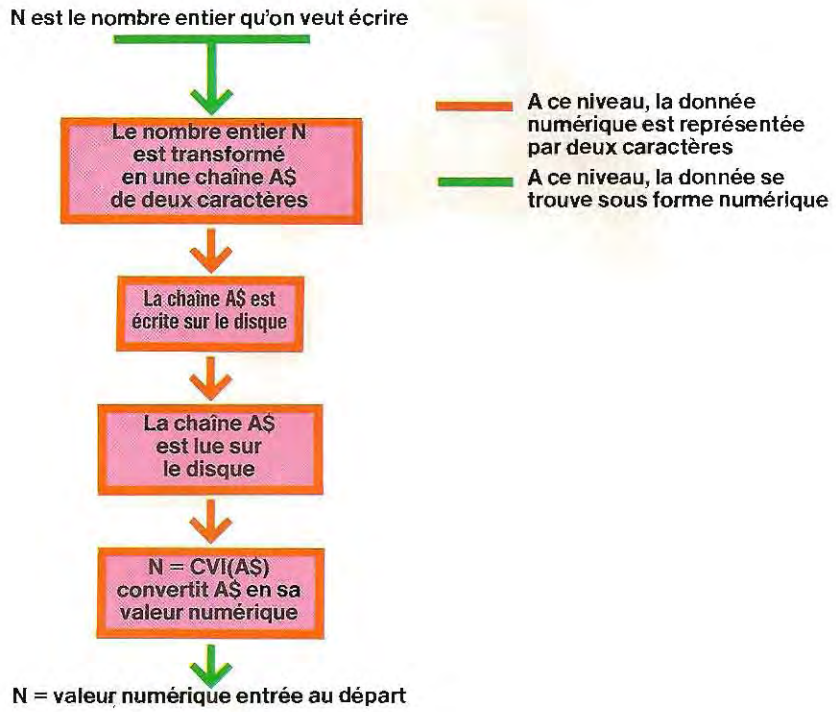
CVI(A$). Convertit une chaîne de deux caractères en son équivalent numérique entier (2 octets). On l'utilise après la lecture de données sur un disque où elles étaient mémorisées sous forme de chaînes. Cette instruction les transforme alors en valeurs numériques et elles devront être retraduites avant sauvegarde.

On trouvera page 454 le schéma décrivant les opérations d'écriture et de lecture de données numériques sur disque dans ce cas.

EMPLOI DE L'INSTRUCTION CHR$



SCHEMA LOGIQUE DE L'ECRITURE ET DE LA LECTURE D'UNE DONNEE NUMERIQUE (ENTIERE) SUR UN DISQUE



CV$(A$). Convertit une chaîne de quatre caractères (4 octets, soit 32 bits) en son équivalent numérique, qui est un réel en simple précision.

CVD(A$). Convertit une chaîne de huit caractères en un réel en double précision.
L'emploi des fonctions CVI, CVS et CVD, ainsi que des fonctions inverses (qui convertissent des réels en chaînes de caractères) est décrit de façon plus détaillée dans le chapitre consacré à la sauvegarde des données sur disque.
Signalons toutefois que ce type de mémorisation, outre ses contraintes, occupe beaucoup plus d'espace qu'une représentation binaire.
Prenons, par exemple, le nombre 12745.
Son écriture en binaire n'utilise que 2 octets (soit 16 bits : 0011000111001001) alors que sa représentation en caractères ASCII occupe 5 octets. En effet, chaque chiffre est alors considéré comme un caractère et mémorisé sur un octet (la chaîne qui représente 12475 est constituée des codes 49, 50, 55, 52 et 53, comme on peut le voir sur la table du code ASCII, page 114).

HEX$(N). Donne une chaîne qui est la représentation hexadécimale (en base 16) de l'entier N. Exemple :

```
10  DEFINT N
20  N = 21
30  E$ = HEX$(N)
```

Ces instructions affectent à la chaîne E$ l'équivalent en base 16 du nombre 21 : soit "15" (1 fois 16 et 5 unités).
Page 455 est listé un programme de démonstration en rapport avec cette instruction et quelques résultats de son exécution.
Cette fonction ne travaille que sur des entiers, mais accepte des réels en argument.
Ceux-ci seront alors convertis automatiquement en entier (ce mécanisme est commun à la plupart des fonctions travaillant sur des

entiers).

INKEY$. Commande la saisie d'un caractère au clavier. C'est donc une instruction d'Entrée/Sortie, et nous étudierons ces fonctions plus en détail dans le chapitre consacré à ce sujet.
Le caractère saisi doit être affecté à une chaîne et la syntaxe exacte est :

C$ = INKEY$

Arrivé à cette instruction, le système indique, par un curseur clignotant, qu'il attend une entrée et suspend l'exécution du programme jusqu'à la frappe d'une touche.

INPUT$(N). Cette fonction est identique à la précédente mais concerne une chaîne de N caractères. Exemple :

C$ = INPUT$(5)

Cette instruction prend les cinq premiers caractères entrés au clavier et les transfère dans la chaîne C$.
On remarquera que INKEY$ n'est qu'un cas particulier de la fonction INPUT$ (c'est le cas où N = 1).

INSTR(N, A$, B$). Permet de déterminer si la chaîne B$ est contenue dans A$.
Le résultat fourni est une valeur numérique (comprise entre 1 et 255) correspondant à la première position où l'on trouve B$.
La valeur 0 indique que la chaîne B$ n'a pas été rencontrée.
Le paramètre N permet de commencer la recherche au N^{e} caractère de A$. S'il est omis, A$ sera inspectée à partir du début.
Les instructions suivantes :

10 A$ = "PRENOM INCONNU"
20 B$ = "O"
30 K = INSTR(A$, B$)

fixent K à 5 puisque la chaîne B$ (uniquement constituée du caractère "O") a été trouvée à la 5^{e} position (5^{e} caractère) de la chaîne A$.
Attention, les espaces sont considérés comme des caractères. Ainsi, la recherche de "U" aurait affecté 14 à K.
Utilisons maintenant cette commande en fixant un paramètre N.
On peut, par exemple, modifier l'instruction 30 comme ceci :

K = INSTR(6, A$, B$)

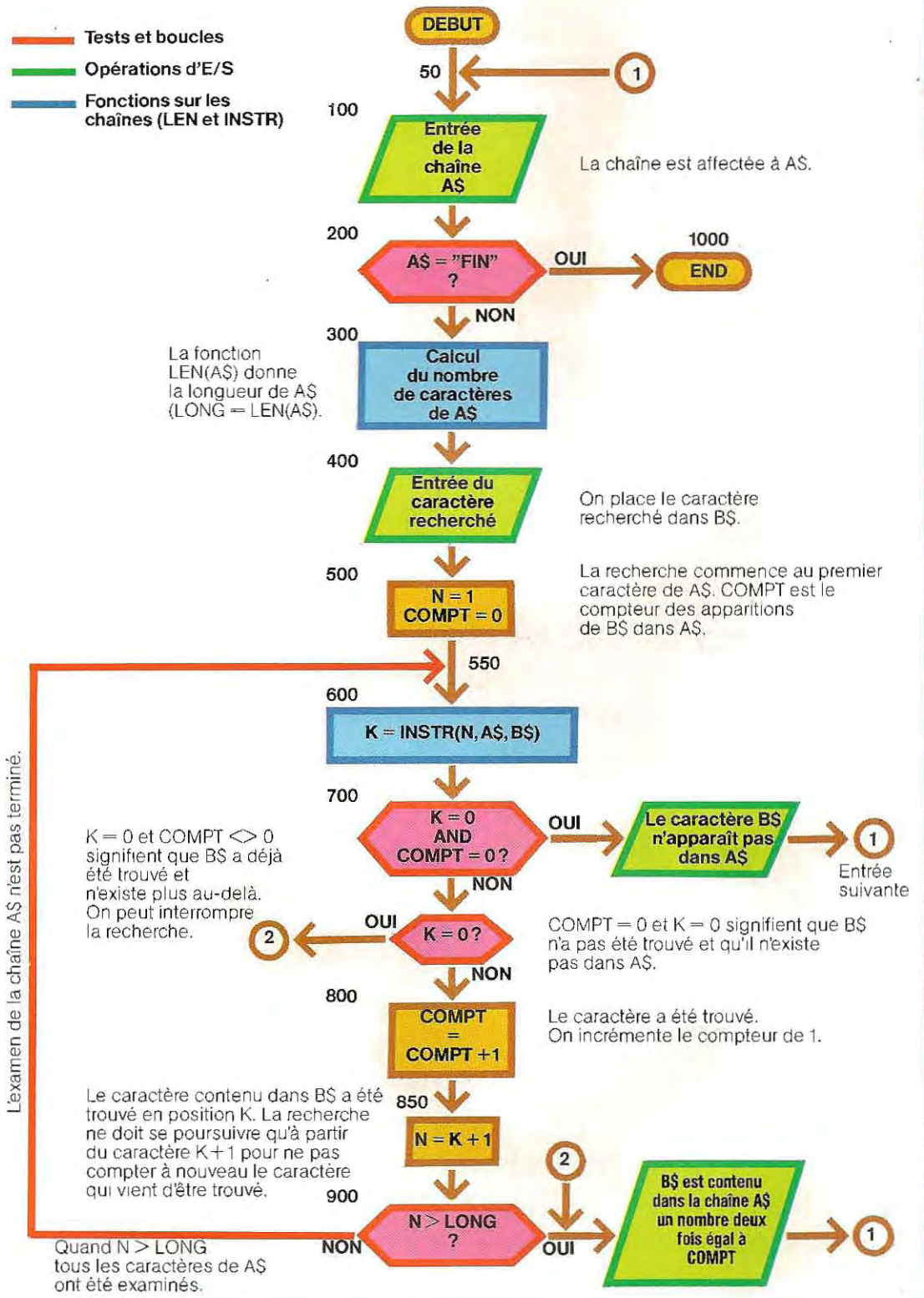
Nous aurons alors K = 11 puisque la recherche a commencé au-delà du "O" de "PRENOM".

EMPLOI DE LA FONCTION HEX$(N)

```
30 DEFINT N
40 BI$=CHR$(27)+"+"+CHR$(7)   ' NETTOYAGE D'ECRAN ET EMISSION D'UN "BIP"
50 '  PREMIER CAS : L'ARGUMENT EST UN ENTIER
60 PRINT BI$
70 PRINT "TAPER (0) POUR SORTIR DE CE PREMIER CAS"
80 INPUT "ARGUMENT : ";N    ' N est défini comme entier à la ligne 30
90 IF N=0 GOTO 130
100 A$=HEX$(N)
110 LPRINT "ARGUMENT : ";N,"VALEUR HEXADECIMALE : ";A$
120 GOTO 80
130 LPRINT
140 ' SECOND CAS : L'ARGUMENT EST UN REEL
145 ' N.B. LA FONCTION HEX$(N) CONVERTIT TOUJOURS LES ARGUMENTS EN ENTIERS
150 PRINT BI$
160 PRINT "TAPER (0) POUR ARRETER LE PROGRAMME"
170 INPUT "ARGUMENT : ";R    ' R est une variable réelle
180 IF R=0 GOTO 220
190 A$=HEX$(R)    ' CONVERTIT R EN HEXADECIMAL ET LE MET DANS A$
200 LPRINT "ARGUMENT : ";R,"VALEUR HEXADECIMALE : ";A$
210 GOTO 170
220 END
```

```
ARGUMENT :   123              VALEUR HEXADECIMALE :   7B
ARGUMENT :   456              VALEUR HEXADECIMALE :   1C8
ARGUMENT :   789              VALEUR HEXADECIMALE :   315
```

ORGANIGRAMME DE COMPTAGE DE L'APPARITION D'UN CARACTERE DANS UNE CHAINE



Page 456, se trouve l'organigramme d'un programme qui effectue les opérations suivantes : il lit une chaîne de caractères saisie au clavier, demande le caractère à rechercher et fournit, en sortie, le nombre de fois où il apparaît. Le programme correspondant, listé sur cette page, utilise la fonction LEN(A$) qui sera étudiée plus tard.

LEFT$(A$,N). Extrait, de la chaîne A$, N caractères à partir de la gauche. Par exemple, la suite d'instructions :

```
10 A$ = "MAURICE RAVEL"
20 B$ = LEFT$(A$,7)
```

affecte à B$ la chaîne "MAURICE", c'est-à-dire les 7 premiers caractères de A$ à partir de la gauche. La valeur de N doit être comprise entre 1 et 255 ; si N est nul, la chaîne B$ sera vide (B$ = "").

Les pages 458 et 459 exposent l'organigramme d'un programme utilisant l'instruction LEFT$ pour la gestion d'un fichier d'adresses.

Les enregistrements du fichier ont la structure suivante :

Champ 1 : Nom et prénom (30 caractères, dont les 20 premiers pour le nom).
Champ 2 : Ville, Numéro, Rue (40 caractères, dont les 10 premiers pour la ville).

Ce fichier, stocké sur disque, contient 1000 adresses ; on demande l'impression de tous les noms d'une ville donnée, ou bien celle de toutes les adresses des personnes de même nom (quelle que soit la ville). Le programme

DETERMINATION DU NOMBRE D'APPARITIONS D'UN CARACTERE DONNE DANS UNE CHAINE

```
1  ' *Calcul du nombre d'apparitions d'un caractère dans une chaîne*
2  ' ***************************************************
3  '
4  '
10 PRINT "POUR SORTIR, ENTREZ LA CHAINE 'SORTIE'"
20 INPUT "Entrez la chaîne de caractères : ";A$
30 IF A$ = "SORTIE" GOTO 200
40 LONG = LEN(A$)
41 '     La variable LONG contient le nombre de caractères de A$
50 INPUT "Caractère à chercher : ";B$
60 N = 1
70 COMPTE = 0
71       ' COMPTE stocke le nombre d'apparitions
72       ' du caractère B$ dans la chaîne A$
80 K = INSTR(N, A$, B$)
90 IF K=0 AND COMPTE=0 GOTO 180    ' on n'a rien trouvé
100 IF K=0 GOTO 140                ' on a fini la chaîne
110 COMPTE = COMPTE+1
120 N=K+1                          ' on entame une nouvelle recherche
130 IF N<LONG GOTO 80              ' à partir du nouveau rang N
140 LPRINT "Le caractère '";B$;"' apparaît"
150 LPRINT COMPTE; "fois dans la chaîne '";A$;"'"
160 LPRINT
170 GOTO 10
171 '
180 LPRINT "Le caractère '";B$;"' n'apparaît pas"
181 LPRINT "dans la chaîne '";A$;"'"
190 GOTO 10
200 END
```

```
Le caractère 'N' apparaît
2 fois dans la chaîne 'VINCENT'

Le caractère 'A' apparaît
2 fois dans la chaîne 'CHANTAL'

Le caractère 'E' apparaît
1 fois dans la chaîne 'ERIC'

Le caractère 'K' n'apparaît pas
dans la chaîne 'VALERIE'
```

doit prendre en compte deux possibilités :

1 / sélection par ville (sur les 10 premiers caractères du champ 2) ;
2 / sélection par nom (sur les 20 premiers caractères du champ 1).

Une partie du listing est représentée page 460 (le module de lecture des données sur disque est omis).

LEN(A$). Cette fonction fournit la longueur de la chaîne A$ (en nombre de caractères), y compris les blancs ou les éventuels caractères spéciaux non imprimables.
L'utilisation de l'instruction LEN est illustrée dans le schéma de la page 456 et en ligne 40 du listing de la page 457.

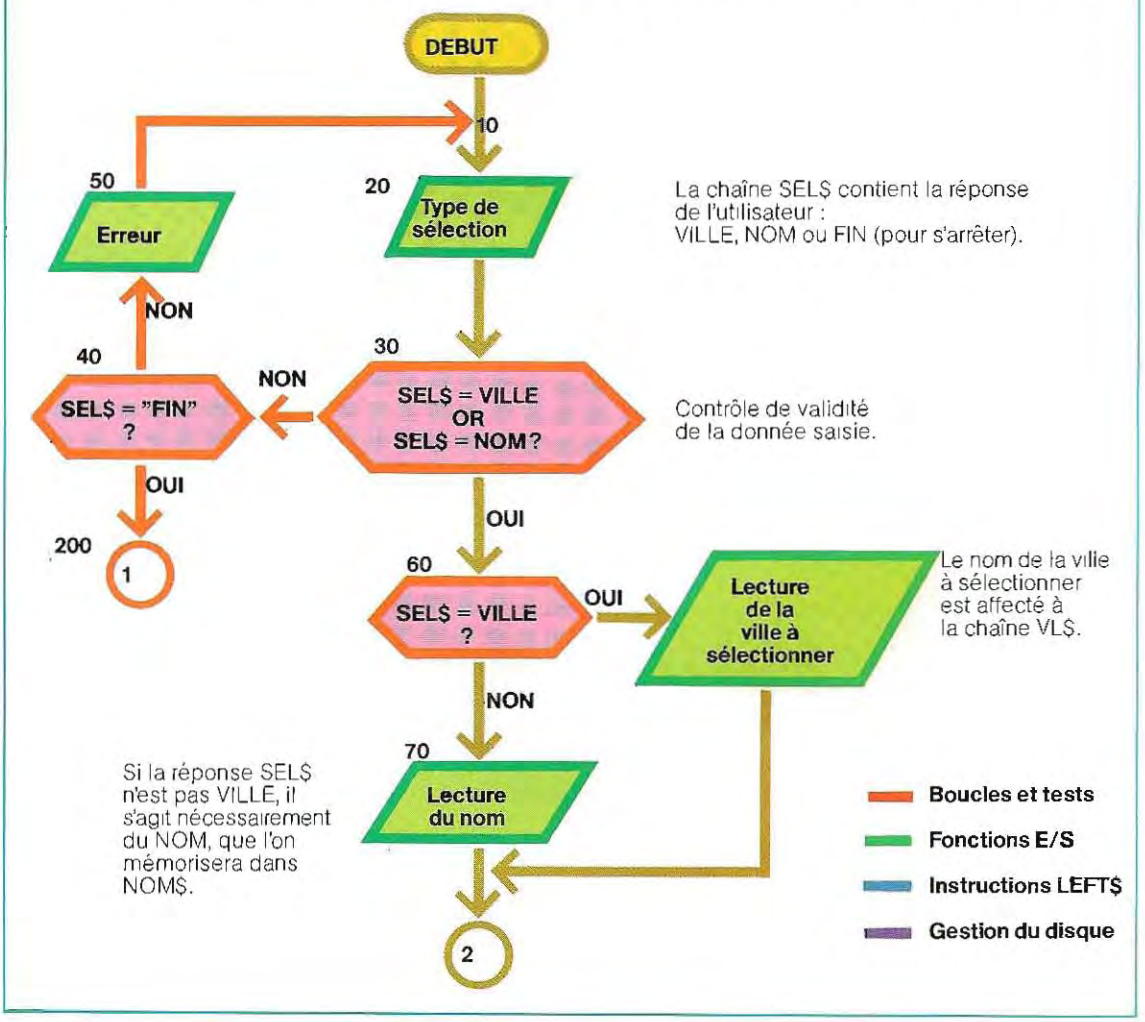
MID$(A$,NS,NC). Extrait de la chaîne A$ un nombre de caractères égal à NC, à partir de la position NS. Par exemple, les instructions :

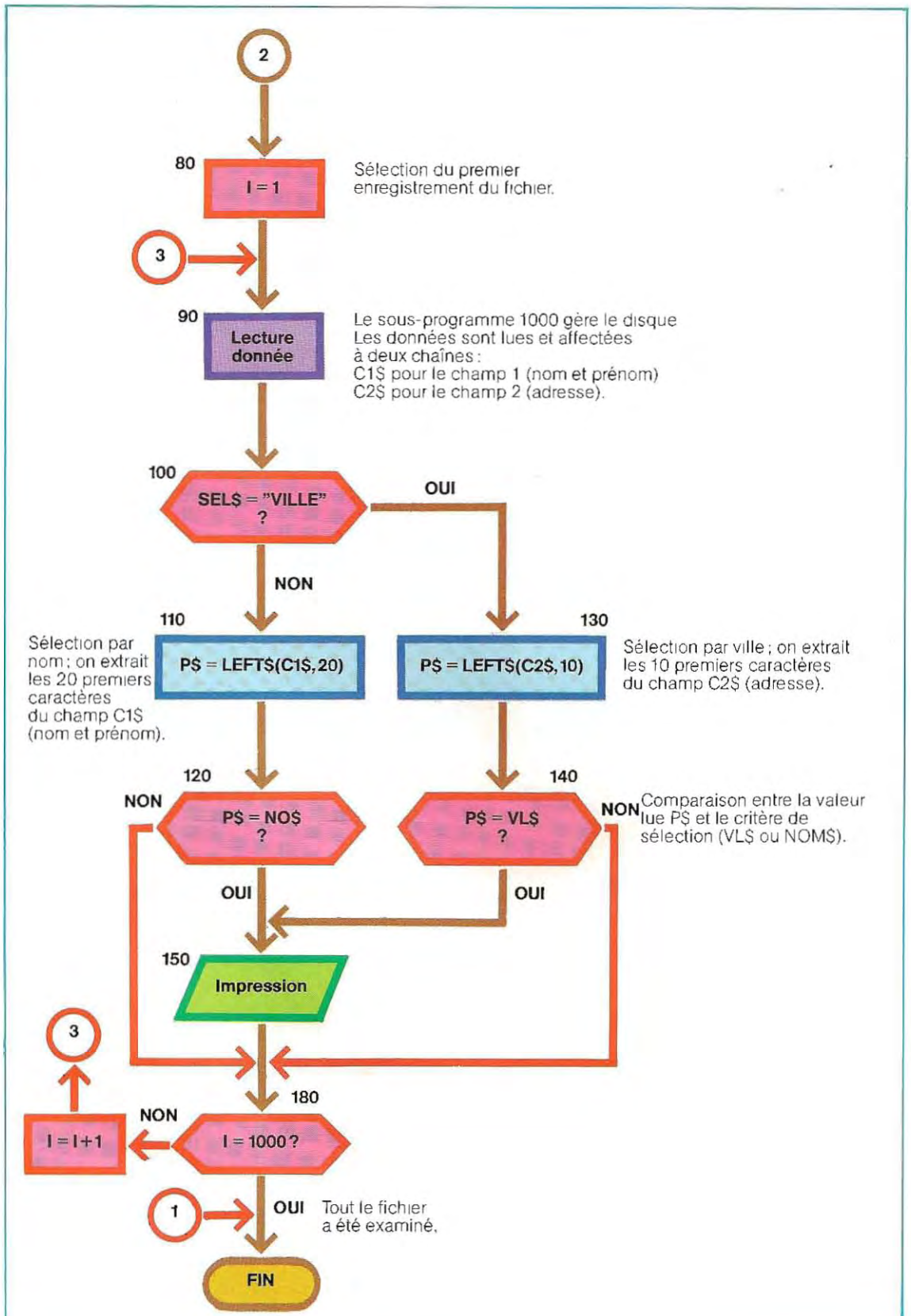
```
10 A$ = "CECI EST UNE CHAINE DE TEST"
20 B$ = MID$(A$,5,7)
```

affectent à B$ sept caractères de A$ à partir du cinquième (de l'espace suivant CECI, jusqu'à la lettre N du mot UNE, les espaces étant comptés comme caractères) ; on aura donc B$ = " EST UN".
L'instruction MID$ peut également effectuer la substitution d'une partie de chaîne. En écrivant :

MID$(A$,NS,NC) = B$

ORGANIGRAMME DE SELECTION DES DONNEES D'UN FICHIER ADRESSES




2
80
I = 1
Sélection du premier enregistrement du fichier.
3
90
Lecture donnée
Le sous-programme 1000 gère le disque
Les données sont lues et affectées à deux chaînes :
C1$ pour le champ 1 (nom et prénom)
C2$ pour le champ 2 (adresse).
100
SEL$ = "VILLE" ?
OUI
NON
110
Sélection par nom ; on extrait les 20 premiers caractères du champ C1$ (nom et prénom).
P$ = LEFT$(C1$,20)
130
P$ = LEFT$(C2$,10)
Sélection par ville ; on extrait les 10 premiers caractères du champ C2$ (adresse).
120
NON
P$ = NO$?
140
P$ = VL$?
NON
Comparaison entre la valeur lue P$ et le critère de sélection (VL$ ou NOM$).
OUI
OUI
150
Impression
3
180
NON
I = I+1
I = 1000 ?
1
OUI
Tout le fichier a été examiné.
FIN

PROGRAMME DE SELECTION DES DONNEES DANS UN FICHIER D'ADRESSES

```
2 REM * Programme de sélection des données dans un fichier ADRESSES *
3 REM ***************************************************
5 REM Exécuté sur EXORSET 33 EN BASICM
6 REM En BASICM, le backslash '\' permet d'insérer un commentaire
7 REM                                          en fin de ligne
10 PRINT "Entrez le critère de sélection, parmi les suivants :"
20 INPUT "VILLE, NOM ou FIN (pour sortir)",SEL$   \ En BASICM, le symbole
30 IF SEL$="VILLE" OR SEL$="NOM" THEN GOTO 70     \ est ',' et non ';'
40 IF SEL$="FIN" THEN GOTO 260
50 PRINT "Erreur : type de sélection non prévu"
60 GOTO 10        \ EN BASICM, on ne met qu'une instruction par ligne
70 IF SEL$="VILLE" THEN INPUT "Ville de sélection",VL$
80 IF SEL$="NOM" THEN INPUT "Nom de sélection",NOM$
90 I=1     \ Sélection du 1er enregistrement
100 GOSUB 300    \ Routine de lecture des données
110 IF SEL$="VILLE" THEN GOTO 170
120 REM Sélection sur le nom
130 P$=LEFT$(C1$,20)     \ extraction du nom (20 caractères)
140 IF P$=NOM$ THEN GOTO 200
150 GOTO 230
160 REM Sélection sur la ville
170 P$=LEFT$(C2$,10)     \ Extraction de la ville (10 caractères)
180 IF P$<>VL$ THEN GOTO 230
190 REM ** Impression des résultats **
200 PRINT "Nom      :";C1$
210 PRINT "Adresse :";C2$
220 REM
230 IF I = 1000 THEN GOTO 260
240 I=I+1
250 GOTO 100     \ boucle sur les enregistrements
260 END
280 REM
290 REM ** Routine de lecture sur disque **
300 REM    (sera détaillée par la suite)
310 RETURN
```

le contenu de la chaîne A$, en partant de la position NS et pour une longueur de NC caractères, est remplacé par le contenu de B$. Par exemple :

```
10 B$ = "75"
20 A$ = "7, rue de Rennes 69"
30 MID$(A$,17,2) = B$
```

donnera : A$ = "7, rue de Rennes 75"
Sur la page ci-contre est listé à titre d'exemple un programme qui utilise les deux formats de cette instruction.

MKI$(N). Convertit un entier (ou une expression entière) N en une chaîne de 2 caractères : c'est la fonction inverse de CVI(A$).

MKS$(R). Convertit un nombre réel R en une chaîne de 4 caractères ; c'est la fonction inverse de CVS(A$).

MKD$(D). Convertit D, réel en double précision, en une chaîne de 8 caractères ; c'est la fonction inverse de CVD(A$).
Le groupe d'instructions MKI$, MKS$, et MKD$ est surtout utilisé pour le transfert sur disque de valeurs numériques dans un format plus compact que l'ASCII, qui transforme chaque chiffre en un caractère. A l'opposé, les opérations de lecture font appel aux fonctions inverses (CVI, CVS, CVD). Ce sujet sera repris et approfondi dans le chapitre consacré aux mémoires de masse.

OCT$(N). Convertit un entier N en une chaîne de caractères correspondant à la représentation octale de N. Après l'exécution des lignes :

```
10 N = 16
20 A$ = OCT$(N)
```

la chaîne A$ contient les caractères 2 et 0 ; 20 est l'équivalent octal du nombre décimal 16. Cette instruction, ainsi que HEX$, a permis de générer les tableaux de conversion des nombres des pages 63 à 67. Le listing du programme correspondant est reproduit page 462.

Certaines instructions, que nous n'avons pas encore étudiées, seront présentées par la suite.

RIGHT$(A$,N). Est le symétrique de LEFT$: on extrait N caractères de la chaîne A$, à partir de la droite. Par exemple, la suite d'instructions :

```
10  A$ = "JEAN MARTIN"
20  B$ = RIGHT$(A$,6)
```

transfère dans B$ six caractères de A$, à partir de la droite ; on obtient : B$ = "MARTIN". Comme avec LEFT$, si N est nul, l'instruction B$ = RIGHT$(A$,0) affecte à B$ la chaîne vide ;

EXEMPLES DE TRAITEMENT DE CHAINES

```
2 REM Programme exécuté sur EXORSET 32 en BASICM
3 REM
4 REM * Extraction d'un groupe de caractères avec MID$ *
6 REM Variables utilisées
7 REM A$ : chaîne initiale
8 REM NS : position du 1er caractère à extraire
9 REM NC : nombre de caractères à extraire
10 INPUT "Chaîne initiale ",A$    \ en BASICM symbole ',' et nom ';'
30 INPUT "Nombre de caractères à extraire      ",NC
40 IF NC = 0 THEN GOTO 240      \ sortie du programme
50 IF NC <= 255 THEN GOTO 80
60 PRINT "Erreur : chaîne trop longue"
70 GOTO 30      \ En BASICM, une seule instruction par ligne
80 INPUT "Position du 1er caractère à extraire ", NS
81 REM Vérification : l'opération demandée est-elle possible ?
85 K = NS+NC-1
90 IF LEN(A$) >= K THEN GOTO 120
95 PRINT "La chaîne '";A$;"' est trop courte"
100 PRINT "pour qu'on manipule ";NC;" caractères à partir du ";NS;"eme"
110 GOTO 20
120 B$ = MID$(A$,NS,NC)
130 PRINT "NS = ";NS;" NC = ";NC
140 PRINT "Chaîne initiale : '";A$;"'"
150 PRINT "Chaîne extraite : '";B$;"'"
151 REM
152 REM * Substitution d'un groupe de caractères avec RIGHT$ et LEFT$
160 INPUT "Caractère(s) de substitution ? ",B$
161 REM Vérification
162 IF LEN(B$) >= NC THEN GOTO 200
170 PRINT "La chaîne '";B$;"' est trop courte"
180 PRINT "Elle doit avoir au moins ";NC;" caractères"
190 GOTO 160
200 C1$ = LEFT$(A$,NS-1)
210 C2$ = RIGHT$(A$,LEN(A$)-NS-NC+1)
215 C$ = LEFT$(B$,NC)
216 A$ = C1$ + C$ + C2$
220 PRINT "Nouvelle chaîne obtenue : '";A$;"'"
225 PRINT
230 GOTO 20
240 END
```

```
NS = 15   NC = 5
Chaîne initiale : 'LE GATEAU EST ENFIN CUIT'
Chaîne extraite : 'ENFIN'
Nouvelle chaîne obtenue : 'LE GATEAU EST ALORS CUIT'

NS = 9   NC = 5
Chaîne initiale : 'IL FAIT CHAUD AUJOURD'HUI !'
Chaîne extraite : 'CHAUD'
Nouvelle chaîne obtenue : 'IL FAIT FROID AUJOURD'HUI !'

NS = 12   NC = 7
Chaîne initiale : 'DONNEZ-MOI L'HEURE'
Chaîne extraite : 'L'HEURE'
Nouvelle chaîne obtenue : 'DONNEZ-MOI LA MAIN'
```

PROGRAMME DE CONVERSION

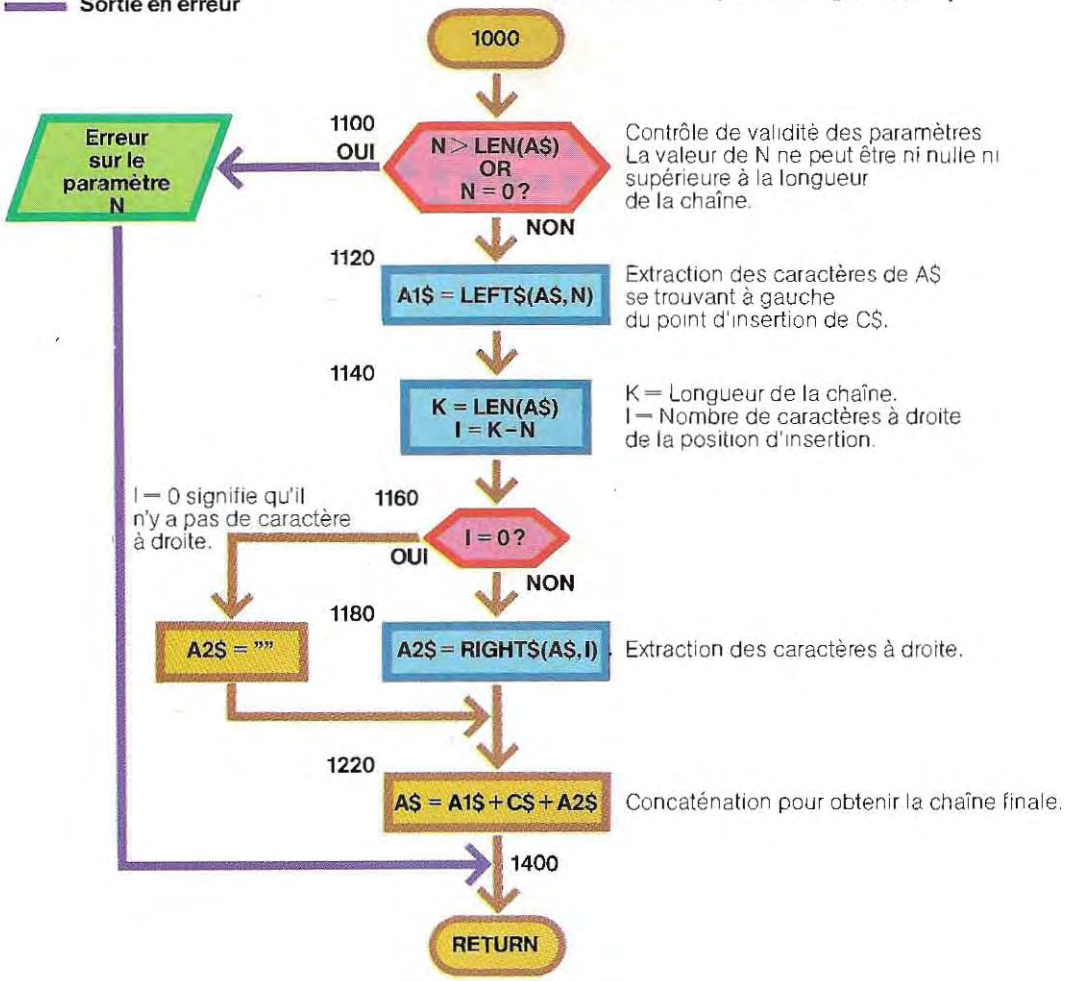
```
15 ' ** DECLARATION DES VARIABLES ET DES TABLEAUX
20 OPTION BASE 1
25 DEFINT A-P
30 DIM B(16)    ' Tableau de travail pour B(16)
35 LPRINT TAB(2);"DECIMAL";TAB(20);"BINAIRE";TAB(50);"OCT";TAB(65);"HEX"
36 LPRINT : LPRINT
40 '
45 A1$ = "valeur incorrecte"  ' test des valeurs entrées
100 ' ** SAISIE DES VALEURS ET CONTROLES
105 '
110 INPUT "valeur initiale";NDEB
115 IF NDEB < 0 OR NDEB > 32000 THEN PRINT A1$ : GOTO 110
120 INPUT "valeur finale";NFIN
125 IF NFIN < NDEB THEN PRINT A1$ : GOTO 120
130 '
135 ' ** Début de boucle. N varie de NDEB à NFIN
140 '
145 FOR N = NDEB TO NFIN
150 '
155 ' ** Conversion en binaire
160 '
165 GOSUB 1000
170 '
175 ' ** Conversion en octal
180 '
185 A3$ = OCT$(N)
190 '
195 ' ** Conversion en hexadécimal
200 '
205 A4$ = HEX$(N)
210 '
215 ' ** IMPRESSION
220 '
225 ' ** Conversion de B(16) en chaîne
230 A2$ = ""
235 FOR I = 16 TO 1 STEP -1
240 A2$ = A2$ + STR$(B(I))
245 NEXT I
275 LPRINT TAB(2);N;TAB(20);A2$;TAB(50);A3$;TAB(65);A4$
285 NEXT N
300 END

1000 '
1005 ' ** Conversion d'un entier en binaire
1010 ' ENTREE : N = nombre entier à convertir
1015 ' SORTIE : B(16) = tableau des bits (symboles 0 et 1)
1020 '                de la représentation binaire de N
1025 '
1030 '
1035 M = N    ' Sauvegarde de la donnée entrée
1040 FOR I = 1 TO 16
1045 B(I) = 0    ' Initialisation
1050 NEXT I
1060 I = 1       ' L'indice I désigne le bit I de B(16)
1065 '             à partir de la droite
1070 D = M/2     ' Division entière (D déclaré de type entier)
1075 K = M - D*2  ' K = reste de la division
1080 B(I) = K    ' Le bit I est à 0 si M est multiple de 2, à 1 sinon
1085 IF M = 1 GOTO 1100   ' Fin de la conversion
1090 M = D
1095 I = I+1 : GOTO 1070  ' On décale d'un bit vers la gauche dans B(16)
1100 RETURN
```

si N est égal à la longueur totale de A$ [N = LEN(A$)], on transfère dans B$ toute la chaîne A$. Page 463 figure l'organigramme de la routine 1000, qui insère un caractère dans une position donnée d'une chaîne ; le listing correspondant se trouve en haut de la page 464. Un autre sous-programme (2000) est ensuite présenté : il permet cette fois de supprimer

ROUTINE 1000 D'INSERTION D'UN CARACTERE DANS UNE CHAINE

Fonctions E/S
Tests
Traitement des chaînes
Sortie en erreur

Données d'entrée du programme.
A$ = Chaîne à modifier
C$ = Caractère à insérer
N = Position d'insertion (en nombre de caractères à partir de la gauche).



Contrôle de validité des paramètres
La valeur de N ne peut être ni nulle ni supérieure à la longueur de la chaîne.

Extraction des caractères de A$ se trouvant à gauche du point d'insertion de C$.

K = Longueur de la chaîne.
I = Nombre de caractères à droite de la position d'insertion.

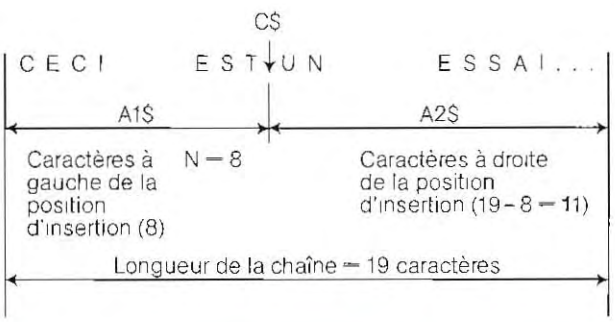
Extraction des caractères à droite.

Concaténation pour obtenir la chaîne finale.

Exemple :

Chaîne saisie = A$ =

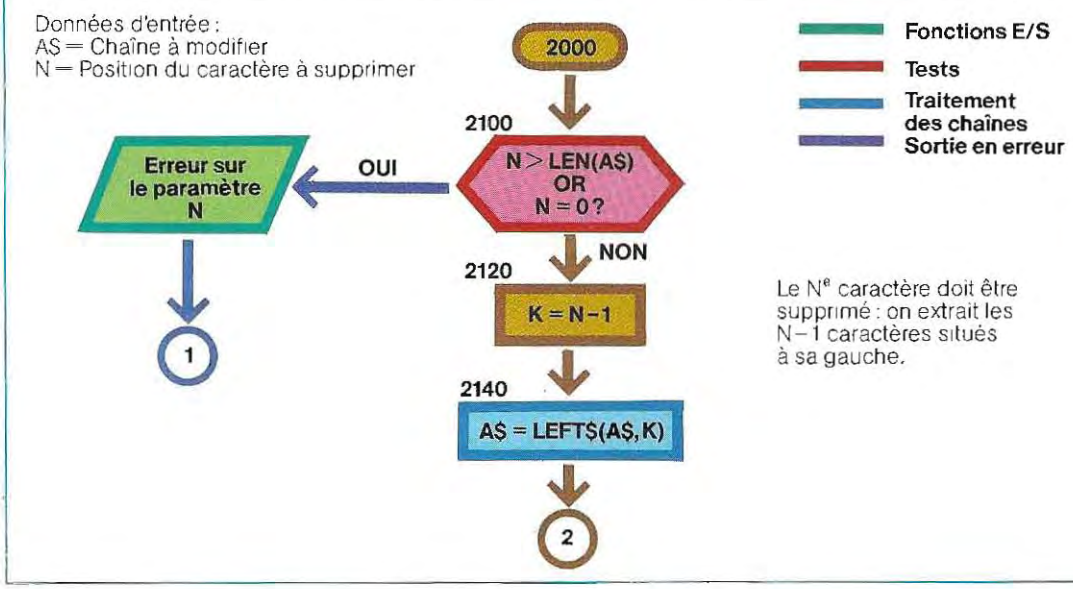
Caractère à insérer :
C$ = blanc

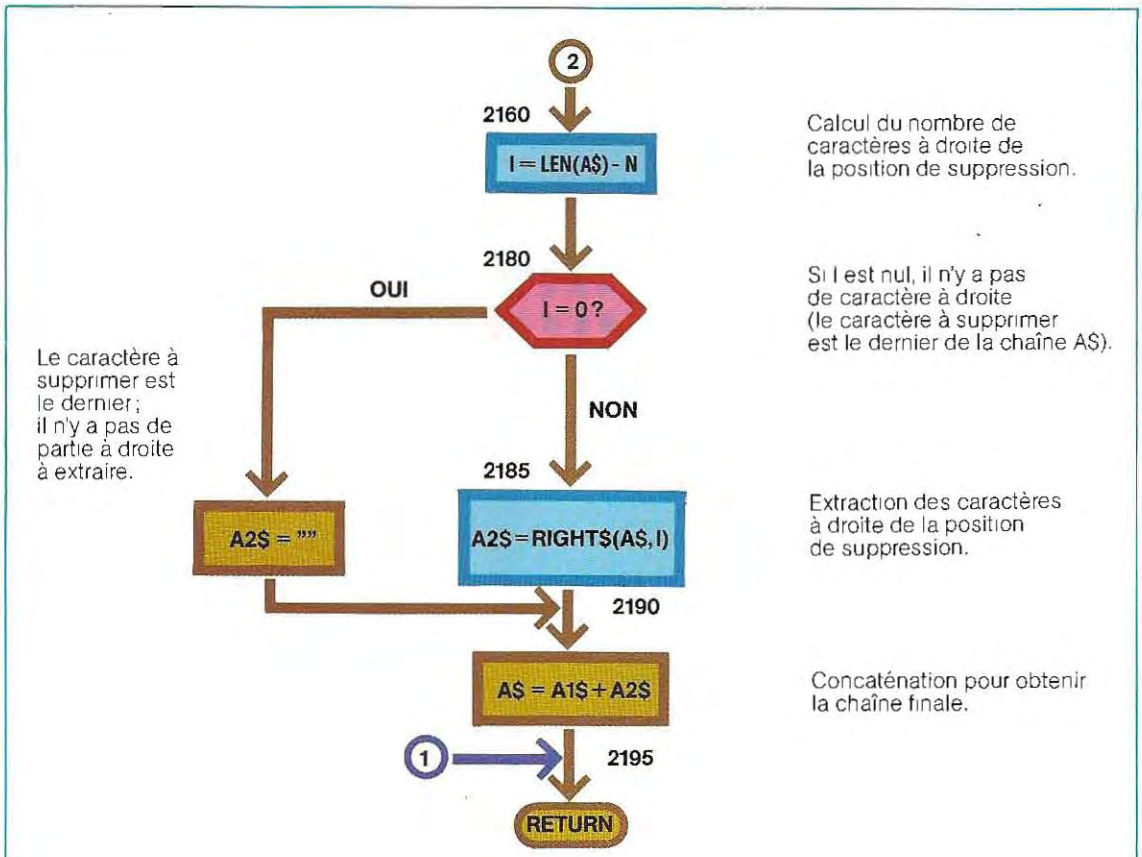


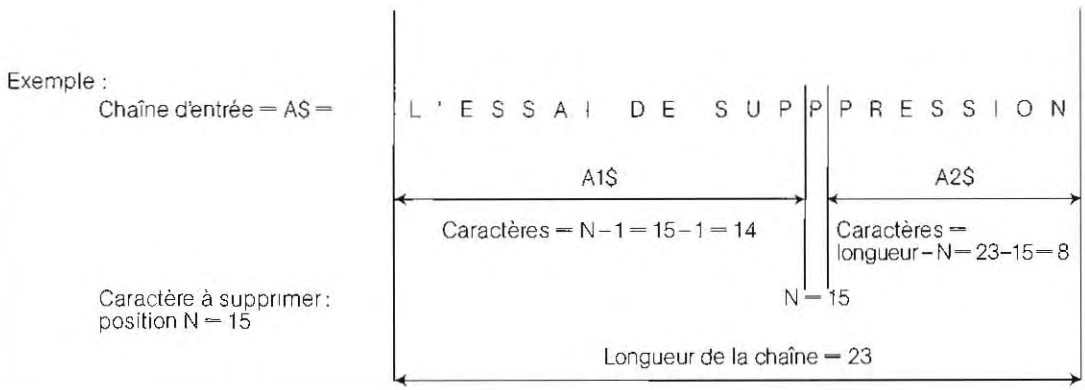
EXEMPLE D'UTILISATION DU SOUS-PROGRAMME 1000

```
10  ' ** PROGRAMME PRINCIPAL **
50  INPUT "CHAINE A MODIFIER"
60  INPUT "CARACTERE A INSERER"
62  ' IF LEN(C$)=0 THEN C$=" "
64  ' SI LE CARACTERE D'INSERTION EST LA CHAINE NULLE,
66  ' L'INSTRUCTION 62 AFFECTE UN BLANC A C$
70  INPUT "QUELLE EST LA POSITION D'INSERTION (nombre de caractères)?"
75  B$=A$ ' SAUVEGARDE DANS B$ DE LA CHAINE D'ORIGINE
80  GOSUB 1000
90  '
100 ' INSTRUCTIONS D'IMPRESSION
110 LPRINT "CHAINE INITIALE :";B$
120 LPRINT "CARACTERE A INSERER (";C$;")"
130 LPRINT "CHAINE MODIFIEE :";A$
132 LPRINT
134 GOTO 50
140 END
1000 ' ** ROUTINE D'INSERTION D'UN CARACTERE DANS UNE CHAINE **
1001 '
1002 '
1003 '
1010 ' DONNEES D'ENTREE DU SOUS-PROGRAMME
1020 '       A$ = CHAINE INITIALE
1030 '       C$ = CARACTERE A INSERER
1040 '       N = POSITION DE L'INSERTION, EN NOMBRE DE CARACTERES
1050 '         A PARTIR DE LA GAUCHE
1100 IF N>LEN(A$) OR N=0 THEN PRINT "ERREUR SUR LE PARAMETRE N"
1120 A1$=LEFT$(A$,N)  ' STOCKAGE DANS A1$ DES CARACTERES A GAUCHE
1140 K=LEN(A$)  ' STOCKAGE DANS K DE LA LONGUEUR DE A$
1145 I=K-N   ' I = NOMBRE DE CARACTERES A DROITE DE LA POSITION D'INSERTION
1160 IF I=0 THEN A2$="":GOTO 1220
1180 A2$=RIGHT$(A$,I)  ' STOCKAGE DANS A2$ DES CARACTERES A DROITE
1220 A$ = A1$+C$+A2$    ' CONCATENATION
1400 RETURN


CHAINE INITIALE : CECI ESTUN ESSAI...
CARACTERE A INSERER ( )
CHAINE MODIFIEE : CECI EST UN ESSAI...
```

ROUTINE 2000 DE SUPPRESSION D'UN CARACTERE DANS UNE CHAINE







(on dit aussi « déléter ») un caractère de rang donné dans une chaîne. Son organigramme se trouve pages 464 et 465, et son listing, suivi d'un exemple d'application, apparaît à la page 466.

Les deux procédures décrites sont programmées à partir des simples fonctions de traitement des chaînes disponibles en Basic. Elles s'avèrent très utiles, car les fonctions d'insertion et de suppression d'un caractère ne figurent pas dans la bibliothèque standard des instructions Basic.

STR$(R). Convertit la valeur numérique R en une chaîne, où chaque caractère représente un chiffre du nombre. Par exemple, les lignes :

```
10  R = 126
20  A$ = STR$(R)
```

EXEMPLE D'UTILISATION DU SOUS-PROGRAMME 2000

```
50 INPUT "CHAINE A MODIFIER (chaine vide pour sortir)";A$
55 IF A$ = "" GOTO 140   ' SORTIE
60 INPUT "POSITION DU CARACTERE A SUPPRIMER"
65 B$ = A$   ' SAUVEGARDE DANS B$ DE LA CHAINE D'ORIGINE
70 GOSUB 2000   ' ROUTINE DE SUPPRESSION D'UN CARACTERE
80 '
90 ' INSTRUCTIONS D'IMPRESSION
100 LPRINT "CHAINE INITIALE:"
110 LPRINT "CHAINE MODIFIEE:"
120 LPRINT
130 GOTO 50
140 END
2000 ' ** SOUS-PROGRAMME DE SUPPRESSION D'UN CARACTERE **
2010 '    DONNEES D'ENTREE
2020 '    A$ = CHAINE A MODIFIER
2030 '    N = POSITION DU CARACTERE A SUPPRIMER
2100 IF N>LEN(A$) OR N=0 THEN PRINT "ERREUR SUR LE PARAMETRE N"
2120 K=N-1
2122 ' K = NOMBRE DE CARACTERES A GAUCHE
2124 '    DE CELUI A SUPPRIMER
2140 A$=LEFT$(A$,K)
2142 ' STOCKAGE DANS A1$ DE CES K CARACTERES
2160 I=LEN(A$)-N
2162 ' I = NOMBRE DE CARACTERES A DROITE DE
2164 '    CELUI A SUPPRIMER
2180 IF I=0 THEN A2$="":GOTO 2190
2185 A2$ = RIGHT$(A$,I)   ' STOCKAGE DANS A2$ DE CES I CARACTERES
2190 A$ = A1$+A2$         ' CONCATENATION
2195 RETURN

CHAINE INITIALE : L'ESSAI DE SUPPPRESSION
CHAINE MODIFIEE : L'ESSAI DE SUPPRESSION
```

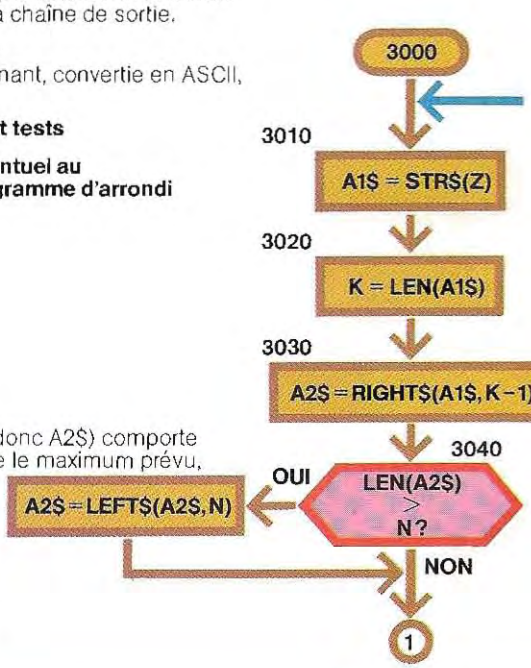
CONVERSION D'UNE VALEUR NUMERIQUE EN ASCII AVEC SUPPRESSION DU PREMIER CARACTERE (SIGNE)

Données d'entrée :
Z = Valeur numérique à convertir en ASCII.
N = Longueur de la chaîne de sortie.

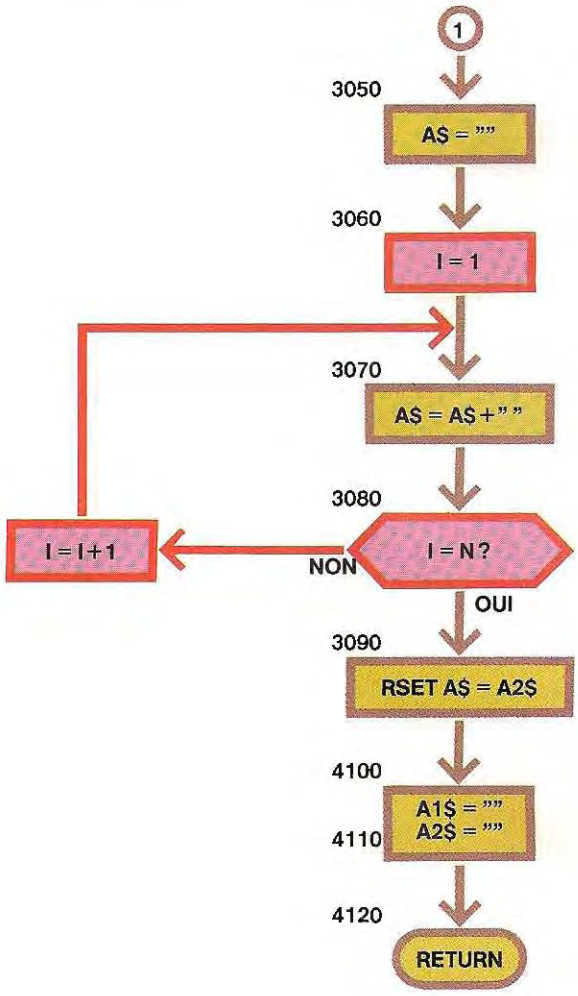
Données de sortie :
A$ = Chaîne contenant, convertie en ASCII, la valeur Z.

Boucles et tests

Appel éventuel au sous-programme d'arrondi



Si le nombre Z (et donc A2$) comporte plus de chiffres que le maximum prévu, il est tronqué. On conserve la partie gauche, de façon à négliger les chiffres les moins significatifs situés à droite (unités).



La chaîne de sortie est initialisée à vide.

Boucle de création de la chaîne A$, de longueur N.

A$ est obtenue en concaténant N fois le caractère espace.

Le contenu de A2$ (nombre codé en ASCII) est cadré à droite dans la chaîne de sortie A$.

Ne servant plus, les chaînes de travail intermédiaires A1$ et A2$ sont réinitialisées à vide

EXEMPLE D'APPLICATION DU SOUS-PROGRAMME 3000

```
50 INPUT "VALEUR NUMERIQUE A CONVERTIR EN ASCII:     ";Z
60 INPUT "LONGUEUR DE LA CHAINE DE SORTIE:     ";N
70 Z1 = Z   ' SAUVEGARDE DANS Z1 DE LA VALEUR INITIALE
80 GOSUB 1000
90 LPRINT "VALEUR NUMERIQUE INITIALE:     ";Z1
100 LPRINT "VALEUR CONVERTIE EN ASCII:     ";A$
110 LPRINT
120 END
3000 ' ** ROUTINE DE CONVERSION D'UN NOMBRE EN ASCII **
3001 '    AVEC SUPPRESSION DU PREMIER CARACTERE
3010 A1$ = STR$(Z)
3020 K = LEN(A1$)
3030 A2$ = RIGHT$(A1$, K-1)
3040 IF LEN(A2$) > N THEN A2$ = LEFT$(A2$,N)
3050 A$=""
3060 FOR I = 1 TO N
3070 A$ = A$ + " "
3080 NEXT I
3090 RSET A$ = A2$
4100 A1$ = ""
4110 A2$ = ""
4120 RETURN
```

Test 14

1 / Certaines des instructions suivantes sont erronées : lesquelles ?
a) A$ = LEFT$(B$,4) b) A$ = MID$(B$,3)
c) A$ = STRING$(3,1) d) A$ = VAL("ABCHD")

2 / Une chaîne contient les 42 caractères suivants :
A$ = "Stylos type A quantité 1250 couleur : rouge"
Sachant que la zone de caractères numériques (1250) débute en position 24 et a une longueur de 4 caractères, écrire un sous-programme réalisant son extraction et sa conversion en valeur numérique.

3 / Écrire un programme qui ajoute une quantité N au nombre obtenu précédemment, et reconstruire la chaîne avec la nouvelle valeur.

4 / Quelles sont les fonctions ou instructions à utiliser pour obtenir :
a) une chaîne de longueur N, contenant N fois le caractère A ;
b) la représentation octale puis hexadécimale d'un nombre N ;
c) la partie entière d'un nombre réel R, obtenue par troncature.

5 / Qu'exécute le programme suivant ?

```
10 A$="*"
20 FOR I=1 TO 20
30 B$= SPACE$(I)
40 PRINT B$;A$
50 NEXT I
```

Les solutions du test se trouvent à la page 473.

transfèrent dans la chaîne A$ les caractères 1, 2, et 6 codés sous forme ASCII. Cette fonction, beaucoup plus utilisée que MK$, MKS$, et MKD$, permet le transfert de valeurs numériques sur disque. Le système d'exploitation utilise une fonction semblable pour afficher un nombre à l'écran.
Les pages 466 (en bas) et 467 présentent l'organigramme d'une routine qui met en œuvre cette fonction, ainsi que le listing du programme correspondant.
Dans ce sous-programme, on a prévu d'éliminer le premier caractère de la chaîne A$. Il représente en effet le signe qui apparaît comme un blanc pour les valeurs positives. Dans beaucoup d'applications, il n'est pas utile de conserver ce signe (toutes les valeurs étant positives), et il convient de récupérer cet espace non significatif.
Dans le cas où l'on voudra conserver le signe, on modifiera le programme localement, en supprimant les instructions 1020, 1030 et 1100, et en récrivant la ligne 1010 (1010 A2$ = STR$(Z). On ne se servira plus de A1$. L'instruction STR$ génère donc une chaîne de longueur variable, en fonction du nombre de chiffres qu'elle doit contenir. On ne peut l'employer directement pour stocker les données en ASCII sur le disque : comme on le verra, elle poserait des problèmes de mise en page (sur listing ou à l'écran), mais surtout de relecture de ces données à format variable.
Dans les programmes d'application, il est préférable d'attribuer toujours la même longueur aux chaînes représentant des valeurs numériques. On définira une longueur maximale des nombres traités, et toutes les chaînes qui les contiendront auront cette même longueur.
Dans l'exemple précédent (routine 1000), la valeur numérique 126 aurait tout d'abord généré une chaîne de 4 caractères (le premier étant un blanc). Pour obtenir un format de 8 caractères (N = 8), on crée une chaîne d'espaces, de longueur N, dans laquelle est transférée (avec cadrage) la chaîne représentant la valeur numérique codée.

L'instruction SPACE$(N) construit automatiquement la chaîne d'espaces voulue. Dans le sous-programme 1000, à titre de démonstration, la chaîne est générée par une boucle.

STRING$(N,K). Crée une chaîne de longueur N, où se répète le caractère correspondant au code ASCII. Par exemple, l'instruction

A$ = STRING$(5,42)

génère la chaîne A$, qui contient cinq fois le symbole * (de code ASCII 42).
Une autre syntaxe est possible :

STRING$(N,B$)

En ce cas, on construit une chaîne de N caractères égaux au premier caractère de B$.

Par exemple, en exécutant les lignes

```
10 B$="***ABC***"
20 A$=STRING$(5,B$)
```

La chaîne A$ obtenue est équivalente à celle de l'exemple précédent : le premier caractère de B$ est en effet représenté par le code 42.

VAL(A$). Fournit la valeur numérique de la chaîne A$. En d'autres termes, c'est la fonction réciproque de STR$(R).
Exemple :

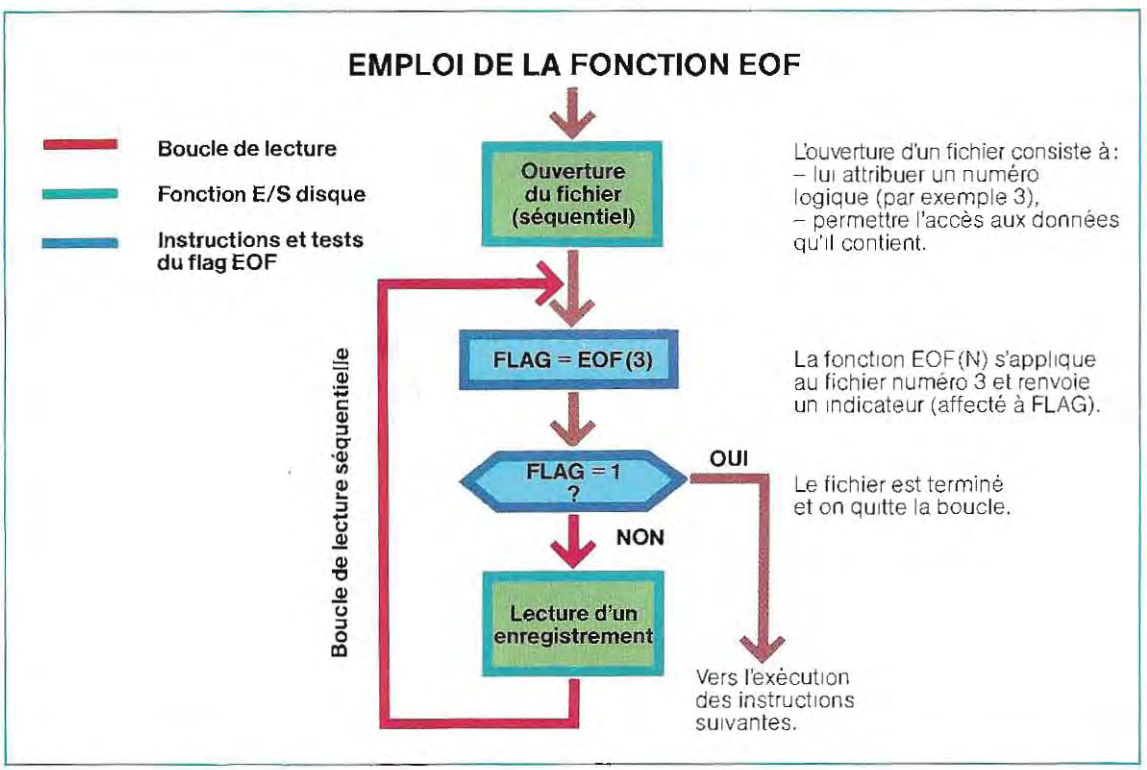
```
10 A$="576.51"
20 R=VAL(A$)
```

Ces instructions transforment le contenu de A$ en valeur numérique et affectent le résultat à la variable R.
On se sert en général de VAL$ après avoir lu des données codées en ASCII sur le disque, afin de les reconvertir en valeurs numériques utilisables par la machine. Notons que l'interpréteur du système réalise une fonction similaire pour saisir un nombre à l'écran.

Fonctions spéciales

A ce groupe appartiennent certaines fonctions particulières gérant les périphériques ou la mémoire. Leur utilisation correcte implique une connaissance approfondie du fonctionnement physique de la machine.

EOF(N). Produit un indicateur qui prend la valeur 1 (ou –1 selon les systèmes) lorsqu'en phase de lecture on rencontre la fin du fichier

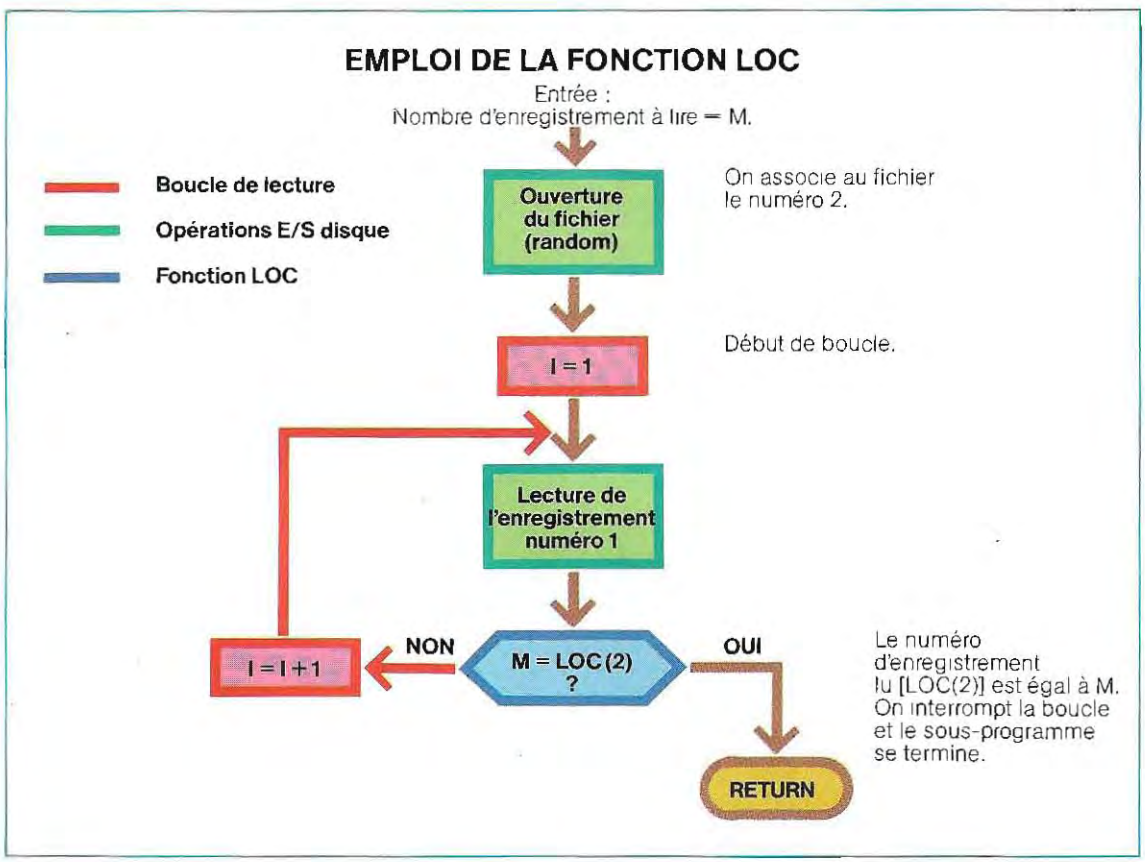


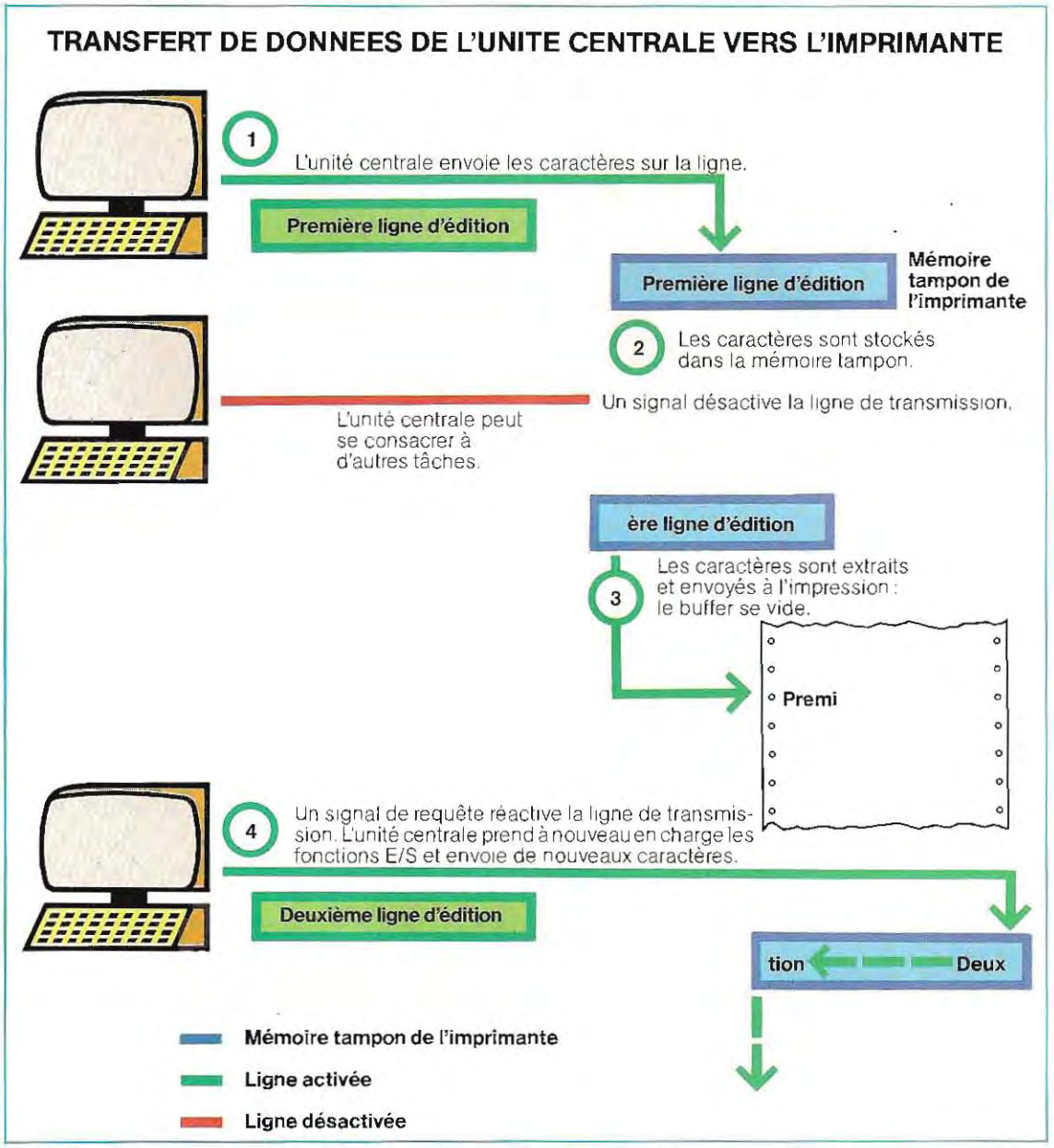
séquentiel N. La valeur N, argument de la fonction, désigne le fichier auquel EOF se réfère. Par exemple, en écrivant EOF(3), on applique la fonction au fichier numéro 3 (ce numéro logique est attribué en phase d'ouverture du fichier). L'organigramme ci-dessous illustre, d'un point de vue logique, la mise en œuvre de la fonction EOF.

FRE(A). Renvoie l'étendue, en nombre d'octets, de la mémoire disponible à un moment donné. L'argument A est « dummy » (fictif) : il n'a aucune signification et n'est mentionné que pour respecter la syntaxe classique d'une fonction.
Ainsi, les instructions FRE(A), FRE(O) ou FRE(C$) seront équivalentes.
Prenons un exemple : supposons une machine ayant 36 K octets de mémoire centrale, où est déjà chargé un programme qui occupe 14 Ko. La fonction FRE(A) fournira la valeur 22000, qui correspond à la place mémoire encore libre.
La syntaxe complète de l'instruction, pour obtenir à l'écran la quantité d'espace mémoire disponible, est PRINT FRE(A).

INP(N). Est une fonction générale d'entrée. Elle renvoie la valeur de l'octet présent sur le « port d'entrée » N. Schématiquement, un port d'E/S peut être représenté par une case mémoire reliée à l'« extérieur » (périphérique). Cette fonction (comme sa « réciproque » OUT) n'est mise en œuvre que pour des applications particulières de gestion des périphériques : l'instruction habituelle permettant la saisie de données au clavier est INPUT.

LOC(N). Retourne le numéro du dernier enregistrement lu ou écrit sur un fichier en accès direct. L'argument N représente le numéro logique attribué au fichier, par exemple en phase d'ouverture.
L'organigramme ci-dessous retrace la lecture d'un nombre déterminé (M) d'enregistrements, à partir d'un fichier en accès direct.
Notre exemple ne constitue pas une application réelle, mais seulement une illustration de l'emploi de la fonction LOC.





LPOS(N) Fournit la position du caractère en cours d'impression à l'intérieur de la mémoire tampon (buffer) de l'imprimante. Comme d'autres périphériques, les imprimantes disposent d'une telle mémoire locale, qui est chargée de stocker les caractères à imprimer. L'unité centrale envoie des caractères au port d'entrées/sorties, qui les transfère dans la mémoire tampon. Lorsque celle-ci est remplie, un signal est envoyé à l'unité centrale, qui se consacre alors à d'autres tâches ; pendant ce temps, l'imprimante extrait les caractères du buffer et les imprime à son propre rythme, son fonctionnement étant totalement indépendant de l'activité de l'unité centrale.
Au fur et à mesure que les caractères seront imprimés, le buffer se videra. Lorsqu'un espace suffisant aura ainsi été libéré, l'imprimante enverra un signal à l'unité centrale, pour demander un nouveau groupe de caractères. Le fonctionnement décrit est schématisé ci-dessus.
La fonction LPOS(N) permet de désynchroniser le fonctionnement de l'unité centrale et de l'imprimante. Ceci prend tout son intérêt si l'on se rappelle que le transfert d'un caractère

(représenté par des signaux électriques) et son impression effective sur papier (qui implique le mouvement d'organes mécaniques), s'effectuent à des vitesses radicalement différentes.
Lorsque LPOS gère les transmissions de données, l'unité centrale* n'est donc pas ralentie par l'imprimante, qui reste le composant le plus lent du système (même avec les modèles les plus sophistiqués comme les imprimantes à laser).

OUT N,M. Cette instruction, présentée au chapitre des fonctions car elle joue le rôle réciproque de INP(M), permet de transférer, vers le port de sortie N, la valeur numérique M. Par exemple :

OUT 3,127

transfère la valeur numérique 127 au port de sortie numéro 3.
Comme INP(N), elle n'est utilisée que dans des cas particuliers : pour les fonctions usuelles d'émission de données, le Basic prévoit d'autres types d'instructions.
L'instruction OUT et la fonction INP sont presque identiques à celles utilisées en langage Assembleur. L'instruction Basic INPUT (qui réalise une lecture de données, comme INP) ne nécessite pas la spécification du périphérique à partir duquel s'effectue la saisie : l'interpréteur Basic (ou le compilateur) désigne systématiquement le clavier pour cette opération. En revanche, en Assembleur (et dans d'autres langages comme le Fortran), cette assignation n'est pas automatique : il appartient au programmeur de spécifier le numéro du port (unité d'entrée) sur lequel doit s'effectuer l'entrée/sortie. Cela explique en partie les raisons qui conduisent à programmer certaines applications en Assembleur plutôt qu'en Basic.
Tous les mini-ordinateurs, ainsi que les micro-ordinateurs modernes, disposent de plusieurs ports d'entrée/sortie et sont dotés d'interfaces spéciales, entre autres pour communiquer par ligne téléphonique avec d'autres ordinateurs. Des programmes en Basic standard ne permettraient pas l'accès à ces dispositifs particuliers.
Les instructions INP et OUT ne s'appliquent pas, elles non plus, à des problèmes particuliers comme la transmission des données, qui mettent en cause la structure hardware de la machine. Le langage le plus adapté sera l'Assembleur, qui reste le plus proche de la machine.
Sur certains micro-ordinateurs de haut de gamme, il existe des instructions utilisables en Basic pour gérer ces interfaces de manière satisfaisante.
De telles machines seront destinées à des applications particulières, comme le contrôle automatique d'appareillages, le recueil de données à l'aide d'instruments, etc.

PEEK(N). Lit et renvoie le contenu de la position N de la mémoire centrale (octet situé à l'adresse N). La valeur retournée est un nombre compris entre 0 et 255. La fonction PEEK requiert donc l'adresse « absolue » de la donnée en question.
Dans tous les langages de haut niveau, en particulier le Basic, chaque emplacement mémoire référencé dans les programmes porte un nom symbolique donné par le système. Aucune adresse physique (position réelle dans l'espace mémoire) n'est donc connue du programmeur ; c'est pourquoi la fonction PEEK, et son instruction réciproque POKE, qui écrit dans la mémoire, sont très peu utilisées dans les programmes d'application classiques.
Cependant, ces fonctions trouvent leur utilité dans la gestion de l'écran vidéo. Si l'on connaît la « cartographie » de la mémoire vidéo, c'est-à-dire la position physique de toutes ses informations, il devient possible de modifier les attributs de l'écran, en changeant, par exemple, la forme du curseur ou sa fréquence de clignotement.

POKE N,M. On a choisi de réserver dans ce chapitre une place à l'instruction POKE, qui opère de manière réciproque vis-à-vis de la fonction PEEK.
L'instruction POKE N,M écrit à l'adresse N de la mémoire, la valeur M (M représentant un octet, est compris entre 0 et 255).

* On mesure la vitesse de l'unité centrale en MIPS (Millions d'Instructions par Seconde). Les vitesses obtenues actuellement sur les machines les plus modernes peuvent atteindre 1 ou 2 MIPS ; la mise en place de techniques particulières, encore à l'étude, permettrait la réalisation de systèmes encore plus rapides.

Solutions du test 14

1 / Les instructions erronées sont :
b) la fonction MID$ requiert deux valeurs entières (par exemple MID$(A$,3,2) ; la première indique la position de début d'extraction, la seconde, le nombre de caractères à extraire ;
d) la fonction VAL, appliquée à une chaîne, renvoie une valeur numérique, qui ne peut être affectée à la chaîne A$.

---

2 / On utilisera la fonction MID$(A$,24,4) pour extraire les caractères numériques et VAL pour les convertir en nombre. Le sous-programme se réduit donc à deux instructions :

```
1000 ' * ENTRÉE DU SOUS-PROGRAMME
1010 B$ = MID$(A$,24,4)
1020 M = VAL(B$)
1030 RETURN
```

---

3 / Ce second sous-programme, plus complexe que le précédent, doit remplacer dans la chaîne le groupe de caractères numériques par le résultat de la somme N+M (N = valeur additionnelle, et M = valeur extraite), lui aussi sous forme de caractères. On scindera A$ en trois : AG$ à gauche, AN$ représentant le nombre converti en chaîne, et AD$ à droite. En concaténant ces trois chaînes, on obtient la nouvelle chaîne A$ complète et contenant la valeur désirée.

```
2000 ' ** ENTRÉE DU SOUS-PROGRAMME
2010 AG$ = LEFT$(A$,23)      ' Partie gauche
2020 AN$ = STR$(N+M)         ' Valeur numérique
2030 AD$ = MID$(A$,28,15)    ' Partie droite
2040 A$ = AG$+AN$+AD$ ' Concaténation
2050 RETURN
```

Puisque l'on a 42 – 27 = 15 caractères à droite de la zone numérique, l'instruction 2030 peut être remplacée par : AD$ = RIGHT$(A$,15).

---

4 / a) STRING$(N,A$), en ayant posé A$ = "A"
b) OCT$(N) et HEX$(N)
c) FIX(R). Attention : la fonction INT(R) n'opère pas par troncature mais par arrondi.

---

5 / La fonction SPACE$(I) crée une chaîne de I caractères blancs (espaces). A l'impression, l'astérisque de A$ se décale à chaque ligne d'une position vers la droite par rapport à la ligne précédente. A la première itération (passe I = 1), on imprime une chaîne ne contenant qu'un seul blanc [B$ = SPACE$(1)], suivie de A$ qui occupe donc la colonne 2. En seconde passe, B$ contient deux blancs et A$ est imprimé en colonne 3, et ainsi de suite.

POS(N). Retourne la position courante du curseur. L'argument N est fictif. Par exemple, la ligne :

```
K = POS(0)
```

affecte à la variable K la position du curseur.

Fonctions définies par l'utilisateur.
On peut enrichir la bibliothèque standard de fonctions Basic en programmant, lors d'une application, de nouvelles fonctions, définies par l'utilisateur.
L'instruction permettant la définition d'une

nouvelle fonction a déjà été étudiée précédemment :

DEF FN NOM = expression

NOM désigne le nom dont on veut baptiser la fonction, et « expression » représente les opérations que la fonction doit effectuer afin de calculer la valeur à renvoyer.
Par exemple, pour évaluer l'hypoténuse d'un triangle rectangle de côtés X et Y, il faut exécuter le calcul suivant (théorème de Pythagore) :

$$H = \sqrt{X^2 + Y^2}$$

qui, une fois traduit en Basic (la racine carrée se calcule par la fonction SQR, l'élévation à une puissance par le symbole Λ), devient :

H = SQR (XΛ2+YΛ2)

Pour exécuter ce calcul plusieurs fois au cours d'un programme, trois possibilités sont envisageables :

1 / écrire chaque fois la formule ;
2 / appeler un sous-programme ;
3 / utiliser une fonction.

La première solution n'est pas rationnelle : elle augmente inutilement l'espace mémoire. Un sous-programme d'une seule ligne (le calcul de H) serait efficace, mais fastidieux : chaque appel nécessite la mention du numéro de ligne où il a été écrit (instruction GOSUB n, avec n = numéro de ligne où commence le sous-programme).
La dernière solution est la plus pratique. En écrivant :

DEF FNA(X,Y) = SQR(XΛ2+YΛ2)

on crée une fonction de nom A, s'appliquant à deux arguments X et Y (côtés), qui exécute le calcul de l'hypoténuse. En mentionnant n'importe où dans le programme :

H = FNA(X,Y)

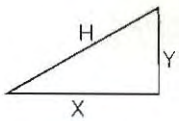
la valeur de l'hypoténuse sera calculée et

Centre de traitement des données équipé de matériel IBM.

K. Reese/Marka

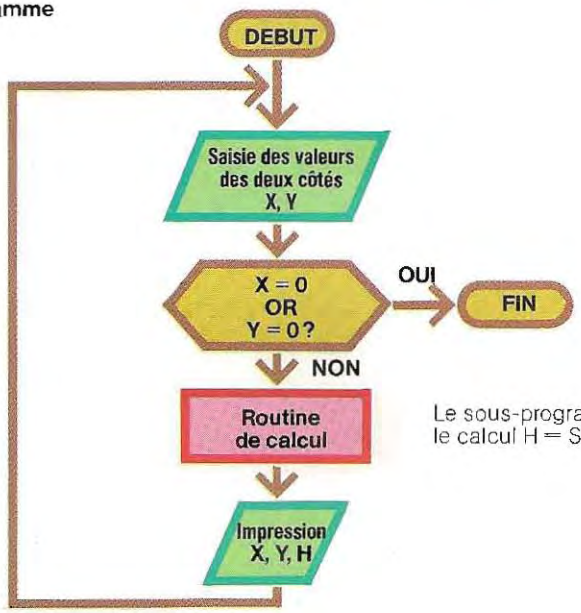
COMPARAISON DES DEUX METHODES DE CALCUL : SOUS-PROGRAMME OU FONCTION

Le calcul à effectuer est
$H = \sqrt{X^2+Y^2}$



Instructions inhérentes à la fonction
Fonctions E/S
Sous-programme

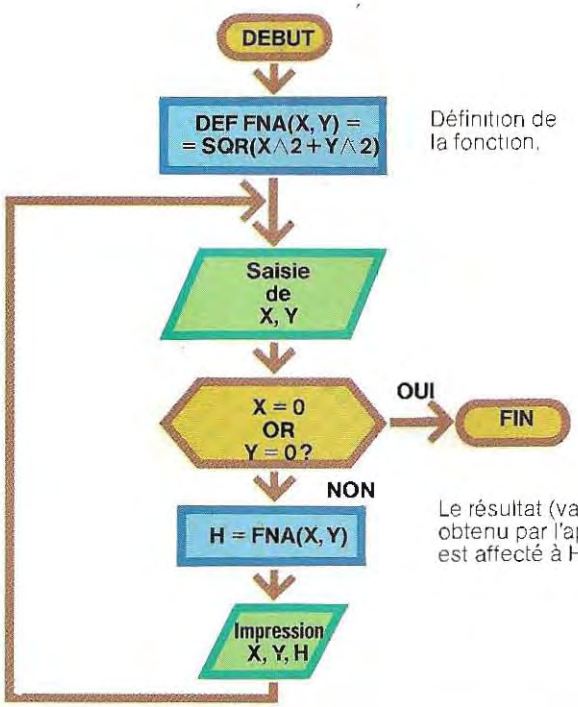
Avec un sous-programme



Le programme se termine lorsqu'un des deux côtés est nul.

Le sous-programme exécute le calcul H = SQR(X<2+Y<2)

Avec une fonction



Définition de la fonction.

Le résultat (valeur de l'hypoténuse), obtenu par l'appel de la fonction, est affecté à H.

affectée à la variable H.
Les organigrammes représentés page 475 correspondent à deux programmes exécutant ce même calcul. Le premier utilise un sous-programme, le second une fonction. Cette page reproduit les listings des programmes ainsi qu'un certain nombre de résultats.
La fonction définie ci-dessus est de type numérique (les arguments et le résultat sont des nombres), mais de manière analogue, on peut envisager des fonctions de chaîne.

PROGRAMME APPELANT UN SOUS-PROGRAMME

```
10 ' ** PROGRAMME UTILISANT UNE SUBROUTINE
20 '
30 ' PROGRAMME PRINCIPAL
40 '
50 PRINT "ENTREZ LES DEUX COTES DU TRIANGLE (X, Y)"
60 INPUT "X =    ";X
70 INPUT "Y =    ";Y
80 IF X = 0 OR Y = 0 GOTO 170
90 GOSUB 200    ' APPEL DU SOUS-PROGRAMME
100 '
110 ' IMPRESSION DU RESULTAT
120 LPRINT "LONGUEUR DU PREMIER COTE X = ";X
130 LPRINT "LONGUEUR DU SECOND COTE Y = ";Y
140 LPRINT "LONGUEUR DE L'HYPOTHENUSE H = ";H
150 LPRINT
160 GOTO 50     ' RETOUR POUR UN NOUVEAU CALCUL
170 END
190 '
190 '
200 ' SUBROUTINE DE CALCUL DE L'HYPOTHENUSE
210 '
220 H = SQR(X*X + Y*Y)    ' FORMULE DE PYTHAGORE
230 '
240 RETURN
```

```
LONGUEUR DU PREMIER COTE     X = 2
LONGUEUR DU SECOND COTE      Y = 2
LONGUEUR DE L'HYPOTHENUSE    H = 2.82843
```

PROGRAMME UTILISANT UNE FONCTION UTILISATEUR

```
10 ' PROGRAMME UTILISANT UNE FONCTION UTILISATEUR
20 '
30 DEF FNA(X, Y) = SQR(X*X + Y*Y)
40 '       L'utilisateur a défini une fonction appelée A(X, Y)
50 PRINT "ENTREZ LES DEUX COTES DU TRIANGLE (X, Y)"
60 INPUT "X =    ";X
70 INPUT "Y =    ";Y
80 IF X = 0 OR Y = 0 GOTO 160
90 H = FNA(X, Y)    ' ON AFFECTE A H LE RESULTAT RENVOYE PAR LA FONCTION
100 ' IMPRESSION DU RESULTAT
110 LPRINT "LONGUEUR DU PREMIER COTE X = ";X
120 LPRINT "LONGUEUR DU SECOND COTE Y = ";Y
130 LPRINT "LONGUEUR DE L'HYPOTHENUSE H = ";H
140 LPRINT
150 GOTO 50     ' RETOUR POUR UN NOUVEAU CALCUL
160 END
```

```
LONGUEUR DU PREMIER COTE     X = 2
LONGUEUR DU SECOND COTE      Y = 2
LONGUEUR DE L'HYPOTHENUSE    H = 2.82843

LONGUEUR DU PREMIER COTE     X = 4
LONGUEUR DU SECOND COTE      Y = 3
LONGUEUR DE L'HYPOTHENUSE    H = 5
```

Un ordinateur sous le capot

L'avènement des microprocesseurs et leur perfectionnement constant résolvent peu à peu tous les obstacles à l'installation, sur une automobile, d'un ordinateur de dimensions compatibles avec les exigences pratiques, et dont la puissance de calcul permet de centraliser l'automatisation complète de toutes les fonctions du véhicule. Le véritable problème est celui de « l'interface » entre l'automobile et son ordinateur.
Afin d'accomplir toutes les fonctions qui lui sont propres, le processeur doit recevoir de façon précise et rapide, les informations, en provenance de l'extérieur, qui sont nécessaires à son intervention finale. Cependant, la variété des tâches à réaliser à bord d'un véhicule et la quantité des paramètres qui les commandent, rendent très complexe le fonctionnement de l'ordinateur.

Un microprocesseur gérant un système d'injection de carburant, par exemple, recevra, afin de fournir un mélange optimal, des informations concernant la vitesse du moteur, son couple, la position des soupapes d'admission, la température et la pression de l'air. Pour chacune de ces valeurs, un ***capteur spécifique*** *sera relié au microprocesseur ; si la conception de ces capteurs n'est pas toujours compliquée – une simple came suffit pour enregistrer la vitesse –, il est évident que leur nombre s'accroît avec l'augmentation des fonctions à contrôler.*
De plus, les informations captées (signaux) doivent être envoyées à l'ordinateur sous la forme adéquate ; autrement dit, toute impulsion enregistrée par le capteur (variation d'une résistance électrique, signaux émis par une cellule photo-électrique, ou toute autre forme d'impulsion) doit être convertie en signal numérique (digital).
Or, l'information issue des capteurs se présente presque toujours sous forme analogique : elle varie en même temps que la quantité mesurée. On obtient ainsi un signal continu qui devra être transformé (échantillonné) en une suite d'informations discrètes (séparées) acceptables par l'ordinateur. Toutefois, avec les techniques actuelles, cette conversion analogique/numérique ne pose plus de problème.

Reste la question de l'interface de sortie, chargée de traduire les décisions de l'ordinateur en commandes numériques appliquées à la voiture. C'est ici qu'une des caractéristiques (sa consommation énergétique très faible) du microprocesseur devient un gros handicap : un signal d'une minuscule fraction d'ampère doit actionner une volumineuse soupape d'alimentation en carburant ou agir sur les freins avec une force suffisante pour arrêter la voiture : il est évident que dans ces cas, le signal doit être considérablement amplifié. Comme en Hi-Fi, on réalise l'amplification en utilisant un petit signal pour en piloter un plus grand. Dans un système en circuit fermé, des oscillations indésirables peuvent alors survenir : une correction, contrebalançant une erreur sans la bloquer et avec un certain retard, peut engendrer une rectification excessive et donc une erreur en sens inverse. Avec un amplificateur Hi-Fi, il se produit alors un bruit aigü ; sur un dispositif de freinage, cela pourrait se traduire par un accident mortel.

Même avec une commande correcte et des oscillations bien amorties, de tels risques sont toujours à redouter. Actuellement, par exemple, les ***freins*** *des voitures sont souvent commandés par des systèmes hydrauliques très sensibles aux oscillations. De plus, dans ce dernier cas, l'interface destinée au contrôle du freinage devra non seulement comprendre un amplificateur, mais aussi un système transformant une impulsion électrique en commande hydraulique.*
Il est évident que les interfaces représentent actuellement le point faible de l'informatisation des voitures, alors que le problème du microprocesseur est déjà résolu : c'est sur la fiabilité des interfaces que porte à l'heure actuelle la majeure partie des recherches dans ce domaine.

Pour ces différentes raisons, l'application des microprocesseurs à l'automatisation des voitures s'est jusqu'ici restreinte à quelques fonctions particulières. Les constructeurs automobiles ont implanté des microprocesseurs dans les modèles les plus sophistiqués de leur gamme, afin d'améliorer leurs performances ou, plus commercialement, à titre d'argument publicitaire. On a assisté de ce fait

à l'informatisation de trois fonctions principales : l'alimentation en carburant, l'allumage et l'équipement du tableau de bord.
Aujourd'hui, certains modèles récents comportent un microprocesseur auquel sont transmises les données telles que la vitesse du moteur, son couple et la position du piston (obtenue à partir de celle de l'arbre à came) ; ces informations constitueront les paramètres d'entrée des programmes stockés initialement dans la mémoire ROM du microprocesseur. Une suite de comparaisons lui permettra de déterminer le moment d'**allumage** optimal, et une interface transmettra les commandes appropriées. Il a déjà été mis en évidence qu'un tel dispositif électronique améliore les performances du moteur, tout en réduisant l'émission des **gaz d'échappement**. Volkswagen, entre autres, a démontré qu'il est possible de maintenir le moteur en régime minimal, en agissant sur la mise en phase.
Toutefois, ces systèmes ne constituent qu'un point de départ : les possibilités d'optimiser les **commandes** par microprocesseur ne se bornent pas là. Depuis les années 1970, certains constructeurs automobiles ont introduit un **capteur de détonation**, afin de rendre les moteurs plus performants en augmentant leur **taux de compression**. A l'heure actuelle, il est rare que les moteurs à essence aient un taux de compression supérieur à 10:1, taux d'ailleurs souvent limité par la teneur en plomb du carburant ; par ailleurs, si ce taux était plus élevé, le moteur serait sujet à **détonation**. Le capteur de détonation est prévenu avant qu'elle ne se produise, par une variation anormale de la vitesse de l'arbre à came, par exemple ; il retarde alors l'allumage. Si l'utilisation du microprocesseur permet déjà une économie considérable de carburant, un taux de compression très élevé et une émission de gaz très peu polluants, il semble que peu de choses puissent être encore améliorées au niveau du moteur lui-même. On reconnaît cependant au microprocesseur des capacités potentielles d'une plus grande portée. Les **soupapes**, par exemple, constituent un vaste champ d'application de la technologie électronique. La société Lucas-CAV a déjà réalisé, en Grande-Bretagne, des soupapes actionnées par des solénoïdes ; depuis 1980, il existe aux Etats-Unis un prototype de moteur doté de soupapes hydrauliques.
Le fonctionnement électronique des soupapes offre de nombreuses possibilités : le conducteur pourrait modifier à volonté les caractéristiques du moteur, en modulant la conduite à l'aide d'un commutateur provoquant une simple variation de la mise en phase et de la durée d'ouverture des soupapes ; inversement, le conducteur pourrait actionner la fermeture simultanée des soupapes, transformant ainsi le moteur en un compresseur assurant un freinage plus efficace. Ce système, selon toute probabilité, permettrait d'améliorer les possibilités de n'importe quel microprocesseur.

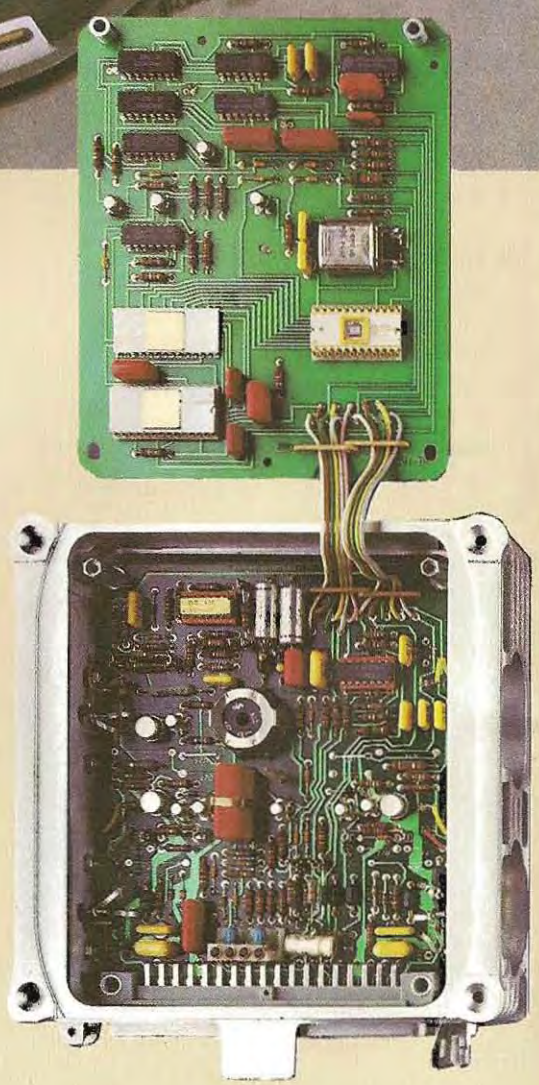
Mais le moteur ne constitue pas la seule cible dans un véhicule. Un autre domaine sollicite l'imagination des ingénieurs mécaniciens : c'est celui des transmissions. On prévoit en effet que dans un avenir proche, le gain de consommation obtenu par avertissement électronique de la transmission sera supérieur à celui qu'on peut attendre des microprocesseurs adaptés au bloc moteur. En concevant un système de ce genre, on programmerait les caractéristiques du moteur dans le microprocesseur, ce qui modifierait automatiquement le **rapport de transmission** en l'adaptant à la vitesse en cours et aux conditions de charge.
Pour confirmer définitivement l'intérêt de cette technique, il faudra probablement attendre les résultats des recherches dans le domaine de la **transmission continue** (CVT). Quoiqu'il en soit, le contrôle par microprocesseur permettra une adaptation plus précise du rapport de transmission en fonction de la vitesse et de la puissance.

Néanmoins, la présence de l'électronique à bord des véhicules « haut de gamme » se manifeste essentiellement sur le tableau de bord : l'apparition en 1976 de la Lagonda, conçue par l'Aston Martin Company, munie de capteurs intégrés (solid state sensor) et d'un tableau de visualisation numérique, a marqué le début d'un engouement pour les véhicules équipés d'un tableau de bord digne d'un scénario de science-fiction. Ces cadrans n'offrent aucun avantage intrinsèque : une représentation « analagique » traditionnelle indique aussi bien, sinon mieux, la

vitesse, le nombre de tours du moteur ou le niveau du carburant. Reconnaissons tout de même cette tentative comme un pas vers la vulgarisation de concepts tels que l'ergonomie et l'optimisation de la conduite. Quelques constructeurs, d'ailleurs, proposent déjà un contrôle continu de la consommation de carburant et de la vitesse moyenne.
Les concepteurs suggèrent de projeter des informations digitales directement sur le pare-brise, afin que le conducteur n'ait jamais à détourner son regard de la route. Si des indicateurs frontaux de ce type (HUD) sont déjà employés depuis quelques années sur les avions de chasse, leur coût de fabrication et les problèmes liés à leur miniaturisation n'ont jusqu'ici pas permis de les transposer aux véhicules de tourisme ; de plus, on n'a pas encore résolu la question de l'éclairage adéquat pour rendre ces cadrans facilement lisibles même par temps ensoleillé.

Un autre domaine, qui n'a pas retenu l'attention méritée, regroupe les systèmes de freinage et de suspension ; toutefois, BMW et Mercedes offrent actuellement sur leurs modèles de haut de gamme un dispositif de freinage antibloquant. Si l'on considère que ces systèmes équipent les avions depuis plus de vingt ans, on peut s'étonner que leur application au domaine automobile ait nécessité tout ce temps : la force d'adhérence qui s'applique aux roues d'un avion est beaucoup plus importante que dans le cas d'un véhicule léger. Le capteur hydromécanique et le système de contrôle des freins devront donc être plus précis et plus rapides (plage de manœuvre plus réduite) dans le cas d'une voiture.
Sur un système électronique anti-blocage, le capteur mesure la force de décélération appliquée à la roue : si elle est trop forte, la roue risque de se bloquer ; le système commande alors un relâchement rapide des freins, puis, aussitôt après, leur réactivation.
Certains ingénieurs pensent que cette technique d'asservissement des freins, à travers un système de tubes hydrauliques, serait transposable à la suspension. Quelques systèmes ont d'ailleurs été mis au point, qui empêchent la voiture d'osciller dans les virages ; mais une solution électronique serait plus efficace. La limitation majeure des techniques traditionnelles de suspension automobile tient au type même de ressort utilisé. Un ressort idéal devrait agir graduellement : ayant une souplesse importante en début de course, de façon à absorber de petites oscillations, il devrait accroître peu à peu sa rigidité, afin de répondre aux fortes sollicitations dont il devient l'objet. Difficile à réaliser avec des ressorts mécaniques, cela devient possible avec des ressorts à forte pression (air liquide) ; un tel système, muni d'un microprocesseur contrôlant une soupape d'évacuation ou de pompage de l'air pourraient assurer une excellente suspension.

Avec l'apparition des microprocesseurs, qui connaissent une importance croissante dans le contrôle de fonctions spécifiques – alimentation en carburant, émission des gaz d'échappement, allumage, transmission, freinage, suspension, direction et instruments de bord –, on peut désormais envisager de réunir tous ces dispositifs, indépendants à l'origine, sous le contrôle d'un ordinateur central. Il y a quelques années, la firme allemande Bosch commercialisa un système de contrôle (Motronic) qui fut installé sur la BMW 732i. L'intérêt du Motronic est qu'il possède un unique processeur pour gérer à la fois l'allumage, l'alimentation en carburant, et de nombreuses autres fonctions optionnelles, grâce à ses interfaces et ses possibilités de programmation externe.
A ses débuts, l'électronique automobile avait tendance à utiliser des processeurs simples, destinés à des fonctions spécifiques. Le VLSI, au contraire, bien qu'encore coûteux, s'avère universel et ouvre le chemin vers une centralisation totale. Un microprocesseur multifonction peut, en effet, être programmé pour contrôler l'allumage, changer de vitesse, allumer automatiquement les phares, etc. En d'autres termes, un seul VLSI accomplira toute une série de tâches, qui requéraient auparavant autant de processeurs spécifiques.
Toutes les fonctions étant asservies par un ordinateur central, il devient plus facile pour une automobile de réagir à la fois rapidement et de manière globale aux stimuli externes, tels que la densité de circulation ou les virages. On pourrait même doter le véhicule d'un radar à micro-ondes relié à l'ordinateur ; la position et la vitesse des autres véhicules,



Maquette de tableau de bord comprenant un cadran d'indication à capteur intégré (à cristaux liquides) et visualisation par réflexion : les données apparaissent sur le pare-brise. A droite : vue intérieure de l'ordinateur de bord.

la présence des piétons, seraient rigoureusement évaluées, et le microprocesseur (même dans le cas d'un brouillard épais), activerait automatiquement le système de freinage, de façon à respecter la distance de sécurité ou à stopper le véhicule si nécessaire. En outre, le radar pourrait donner au conducteur la possibilité d'effectuer un dépassement sur route sinueuse, en permettant de « voir » au-delà des virages.

Dès la fin des années 1970, des constructeurs allemands ont expérimenté des systèmes à ordinateur de bord recueillant, depuis la route, les informations sur les conditions de circulation et les parcours conseillés, pour les visualiser sur un cadran à l'intérieur de l'automobile. Il n'est pas utopique de penser que de tels systèmes de « conduite assistée par ordinateur » seront commercialisés dans un avenir proche.

Smiths Industries Ltd.

Par exemple, la fonction :

DEF FNA$(C$,N) = LEFT$(C$,N) + D$

prélève les N caractères les plus à gauche de C$ et y ajoute la chaîne D$. On trouvera ci-dessous et page 482 l'organigramme et le listage d'un programme utilisant la fonction ainsi définie.

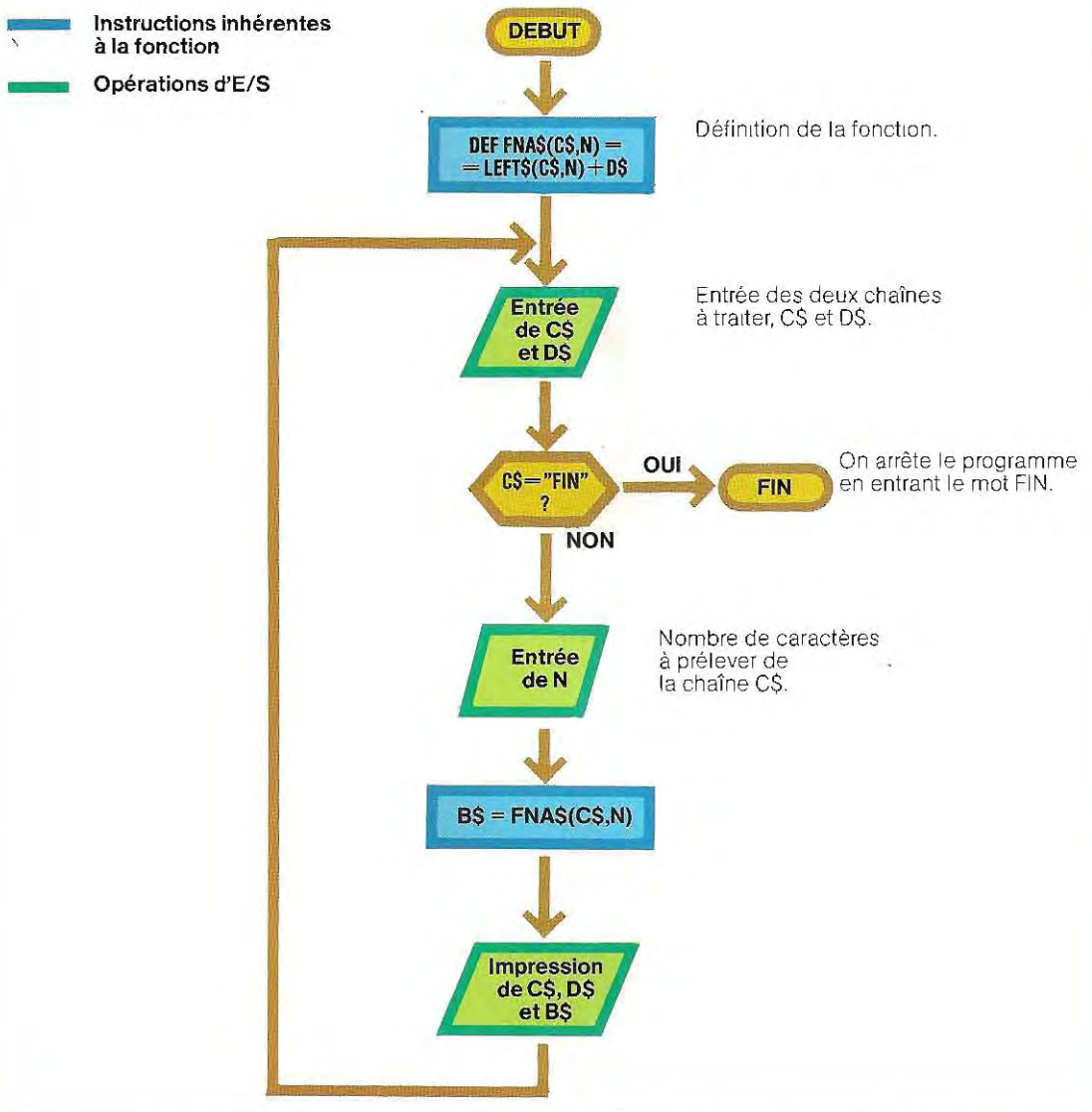
Nous allons maintenant développer deux applications d'usage général relatives aux fonctions numériques, d'une part, et au traitement des chaînes, d'autre part.

Fonctions numériques : résolution d'équations du second degré. Ces équations se présentent sous la forme :

$$A \times X^2 + B \times X + C = 0 \quad .$$

où A, B et C sont des paramètres fixes. Quelle que soit la valeur de ces paramètres, il existe toujours deux « racines » (valeurs particulières de X) pour lesquelles la partie gauche

TRAITEMENT DE CHAINES PAR UNE FONCTION PREDEFINIE



PROGRAMME DU TRAITEMENT DE CHAINES

```
10  ' ** PROGRAMME DE DEMONSTRATION : FONCTION DE TRAITEMENT DE CHAINES **
20  '
30  '                          PROGRAMME = FN/CHAIN
40  '
50  DEF FNA$(C$,N) = LEFT$(C$,N)+D$
60  PRINT "TAPEZ LES DEUX CHAINES (C$ et D$) A TRAITER"
70  PRINT
80  PRINT "POUR SORTIR DU PROGRAMME TAPEZ - FIN -"
90  INPUT "C$ = ";C$
100 INPUT "D$ = ";D$
110 IF C$ = "FIN" GOTO 280
120 PRINT "TAPEZ LE NOMBRE DE CARACTERES A PRELEVER DANS C$"
130 INPUT "N = ";N
140 B$ = FNA$(C$,N)
150 ' La ligne 140 affecte à la variable B$ le résultat
160 ' de la fonction définie en ligne 50
170 '
180 ' INSTRUCTIONS D'IMPRESSION
190 '
200 LPRINT "CHAINES A TRAITER (C$ et D$)"
210 LPRINT C$
220 LPRINT D$
225 LPRINT "NOMBRE DE CARACTERES PRELEVES DANS C$ = ";N
230 LPRINT
240 LPRINT "CHAINE TRAITEE"
250 LPRINT B$
260 LPRINT
270 GOTO 60        ' RETOUR POUR UN NOUVEAU CALCUL
280 END
```

```
CHAINES A TRAITER (C$ et D$)
ALPHABETIQUE
NUMERIQUE
NOMBRE DE CARACTERES PRELEVES DANS C$ = 5

CHAINE TRAITEE
ALPHANUMERIQUE
```

de l'égalité s'annule effectivement. Ces valeurs sont :

$$X1 = (-B+\sqrt{B^2-4\times A\times C})/(2\times A)$$
$$X2 = (-B-\sqrt{B^2-4\times A\times C})/(2\times A)$$

Le programme calcule les racines en utilisant deux fonctions définies selon ces formules.

FNX1(A,B,C)=(−B+SQR(B∧2−4*A*C))/(2*A)
FNX2(A,B,C)=(−B−SQR(B∧2−4*A*C))/(2*A)

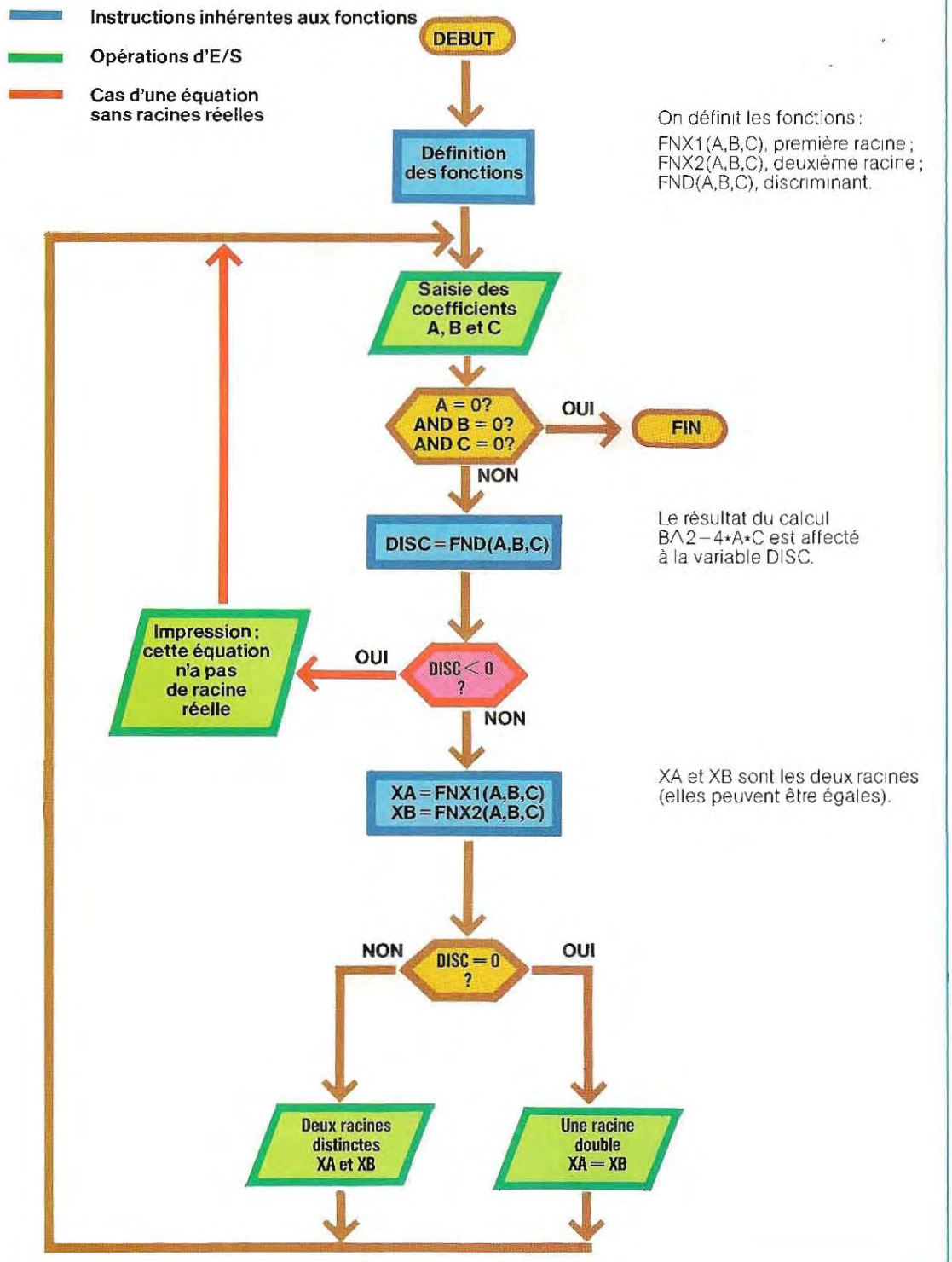
L'utilisation de l'instruction SQR peut toutefois poser un problème quand son argument prend une valeur négative. En effet, le signe de l'expression : $B^2 - 4\times A\times C$ varie selon les valeurs de A, B et C.
Cette expression, appelée **discriminant** de l'équation, nous oblige donc à distinguer trois cas :

premier cas : le discriminant est positif et sa racine carrée (SQR) appartient à l'ensemble des réels. Il existe alors deux racines à l'équation ;
deuxième cas : le discriminant est nul (et sa racine carrée aussi). Les deux racines sont donc égales (on parle de « racine double ») ;
troisième cas : le discriminant est négatif et sa racine carrée est un nombre complexe et non réel. X1 et X2 n'existent donc pas dans l'ensemble des réels.
Avant d'effectuer les opérations calculant les racines de l'équation, on s'assurera donc de la valeur du discriminant pour savoir dans quel cas on se trouve. La fonction à définir pour déterminer cette valeur est :

DEF FND(A,B,C) = B∧2 − 4 * A * C

L'organigramme de ce programme est présenté page 483 et son listage page 484.

RESOLUTION D'EQUATIONS DU SECOND DEGRE



RESOLUTION D'EQUATIONS DU SECOND DEGRE

```
10 ' ** RESOLUTION D'EQUATIONS DU SECOND DEGRE **
20 '
30 '                    PROGRAMME = EQUAT. 2
40 '
50 ' FONCTIONS DEFINIES PAR L'UTILISATEUR
60 DEF FNX1(A,B,C)=(-B+(SQR(B*B-4*A*C)))/2*A)
70 DEF FNX2(A,B,C)=(-B-(SQR(B*B-4*A*C)))/2*A)
80 DEF FND(A,B,C)=B*B-4*A*C
90 '
100 PRINT "ENTREZ LES COEFFICIENTS DE L'EQUATION (A, B et C)"
110 INPUT "A=";A
120 INPUT "B=";B
130 INPUT "C=";C
140 '
150 IF A=0 AND B=0 AND C=0 GOTO 460        ' EQUATION NULLE POUR SORTIR
160 DISC = FND(A,B,C)        ' On affecte à la variable DISC le résultat
170 '                          de la fonction FND(A,B,C) qui calcule
180 '                          le discriminant
190 IF DISC<0 THEN GOTO 451
200 XA=FNX1(A,B,C)        ' XA et XB SONT LES DEUX RACINES
210 XB=FNX2(A,B,C)
220 IF DISC=0 THEN GOTO 240 ELSE GOTO 400
230 '
240 ' PREMIER CAS: DISCRIMINANT=0, LES DEUX RACINES SONT IDENTIQUES
250 LPRINT "COEFFICIENTS DE L'EQUATION AX²+BX+C"
260 LPRINT "A=";A,"B=";B,"C=";C : LPRINT
270 LPRINT "DISCRIMINANT=0 : IL Y A DEUX RACINES REELLES EGALES"
280 LPRINT "XA = XB = ";XA
290 LPRINT : LPRINT
300 GOTO 100                ' RETOUR POUR UN NOUVEAU CALCUL
400 '
410 ' DEUXIEME CAS : DISCRIMINANT POSITIF, IL Y A DEUX RACINES REELLES ET DISTINCTES
415 LPRINT "COEFFICIENTS DE L'EQUATION AX²+BX+C"
420 LPRINT "A=";A,"B=";B,"C=";C : LPRINT
422 LPRINT "DISCRIMINANT = ";DISC
424 LPRINT "DISCRIMINANT POSITIF : IL Y A DEUX RACINES REELLES ET DISTINCTES
426 LPRINT
428 LPRINT "RACINES"
430 LPRINT "XA = ";XA
440 LPRINT "XB = ";XB
445 LPRINT : LPRINT
450 GOTO 100                ' RETOUR POUR UN NOUVEAU CALCUL
451 LPRINT "COEFFICIENTS DE L'EQUATION AX²+BX+C"
452 LPRINT "A=";A,"B=";B,"C=";C : LPRINT
453 LPRINT "DISCRIMINANT = ";DISC
454 LPRINT "L'EQUATION N'A PAS DE SOLUTION DANS L'ENSEMBLE DES REELS"
455 LPRINT "ELLE ADMET POUR SOLUTION DES NOMBRES COMPLEXES"
456 LPRINT : LPRINT
457 GOTO 100
460 END
```

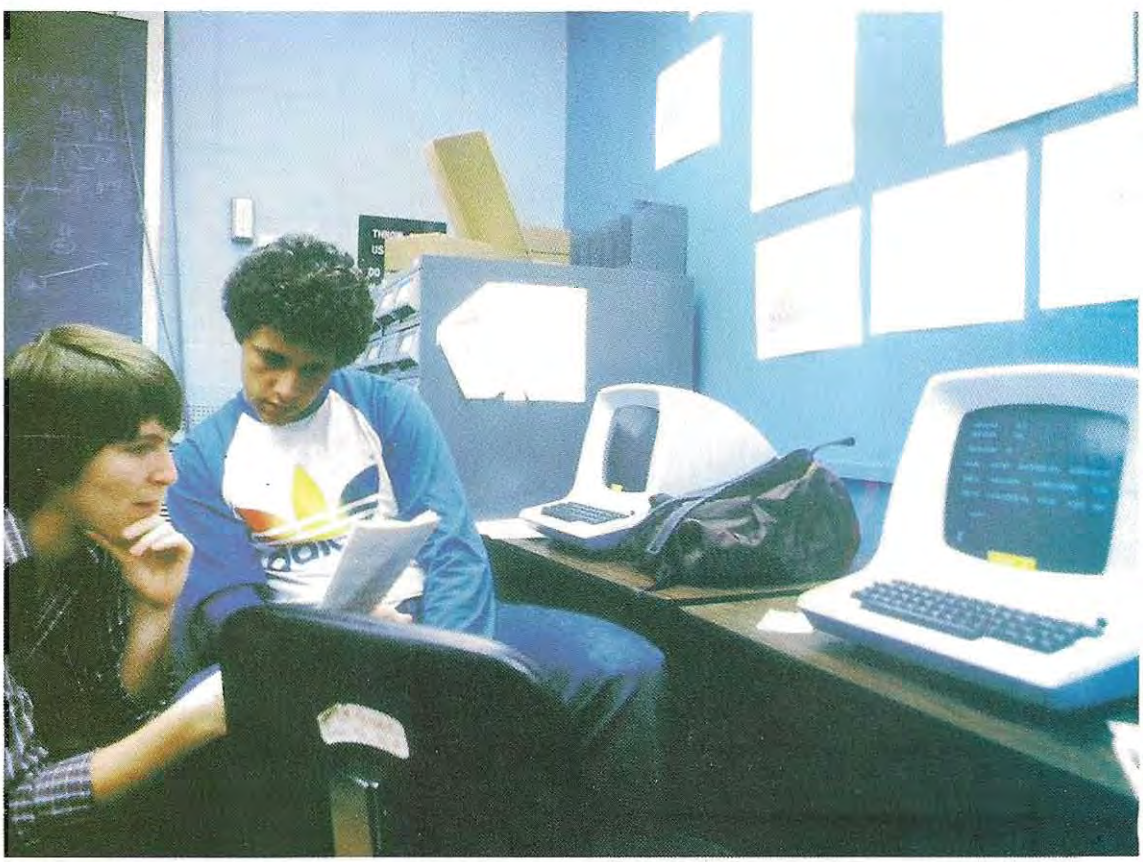
```
COEFFICIENTS DE L'EQUATION AX²+BX+C
A = 2       B = 3       C = 60      ''

DISCRIMINANT = -471
L'EQUATION N'A PAS DE SOLUTION DANS L'ENSEMBLE DES REELS
ELLE ADMET POUR SOLUTION DES NOMBRES COMPLEXES


COEFFICIENTS DE L'EQUATION AX²+BX+C
A = 3       B = -10       C = 1

DISCRIMINANT = 88
DISCRIMINANT POSITIF : IL Y A DEUX RACINES REELLES ET DISTINCTES

RACINES
XA = 3.23014
XB = .103195
```

Fonctions sur chaînes: contrôle des minuscules et des majuscules. Certains programmes traitent principalement des textes ou des fragments de textes (fichiers). L'usage des majuscules en début de phrase ou de nom propre peut alors être impératif.
Un sous-programme assurant le contrôle et l'inversion éventuelle (majuscule/minuscule) permet de soulager l'opérateur lors de l'entrée des données. C'est la machine elle-même qui transforme les lettres lors de la frappe.
On trouvera pages 486 et 487 l'illustration (organigramme et listing) d'un tel programme, qui pourrait prendre place dans un traitement de fichiers d'adresses, par exemple.
Les fonctions utilisées sont:

FNP$(I) = MID$(A$,I,1)
(extraction du premier caractère de la chaîne A$)

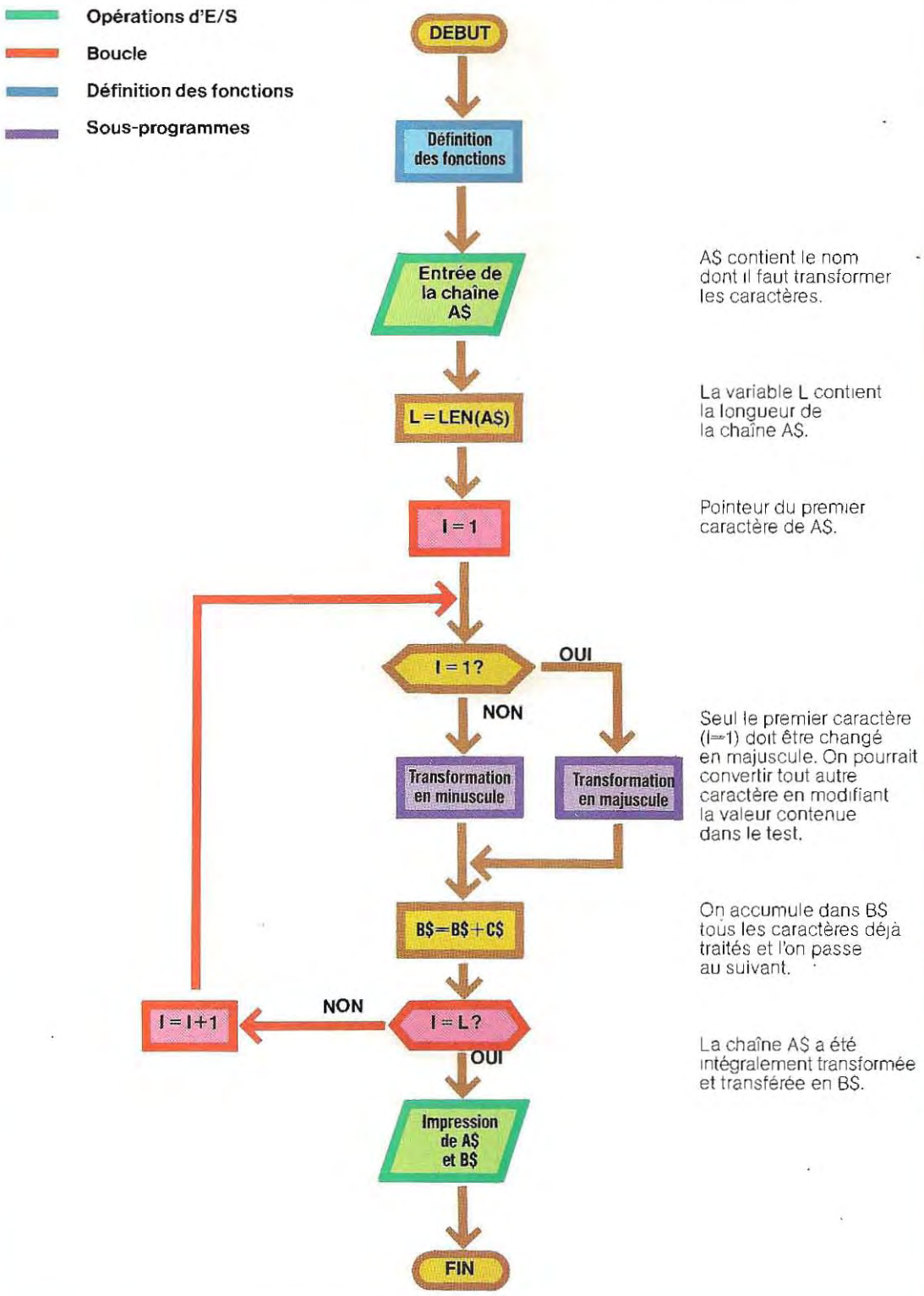
FNV$(V) = CHR$(V – 32)
(passage de minuscule en majuscule)

Aux Etats-Unis, l'enseignement fait de plus en plus appel à l'ordinateur. Sur la photo, des étudiants de la Georgetown University.



J. Pickerell/Marka

PROGRAMME DE CONTROLE ET DE TRANSFORMATION DE CARACTERES



PROGRAMME DE CONTROLE ET DE TRANSFORMATION DE CARACTERES

```
10 '
20 ' ** PROGRAMME DE CONTROLE ET DE TRANSFORMATION DE CARACTERES **
30 '
40 '                    PROGRAMME = MAJ/MIN
50 ' FONCTIONS DEFINIES PAR L'UTILISATEUR
70 DEF FNP$(I)=MID$(A$,I,1)     ' EXTRACTION DU CARACTERE DE RANG(I)
80 '                              DANS LA CHAINE A$
90 DEF FNV$(V)=CHR$(V-32)       ' PASSAGE DE MINUSCULE EN MAJUSCULE
100 DEF FNZ$(V)=CHR$(V+32)      ' PASSAGE DE MAJUSCULE EN MINUSCULE
110 '
120 INPUT "TAPEZ LA CHAINE A TRAITER ";A$
130 L=LEN(A$)           ' LA VARIABLE L CONTIENT LA LONGUEUR DE LA CHAINE A$
140 FOR I=1 TO L
150 IF I=1 THEN GOSUB 1000 ELSE GOSUB 2000
160 '
170 ' LES ROUTINES DE TRANSFORMATION EN MAJUSCULE ET EN MINUSCULE
180 ' COMMENCENT RESPECTIVEMENT EN 1000 ET EN 2000
190 '
200 B$=B$+C$            ' UNE FOIS TRAITES, LES CARACTERES SONT STOCKES
210 '                     UN PAR UN DANS B$
220 NEXT I
230 '
240 ' INSTRUCTIONS D'IMPRESSION
250 LPRINT "CHAINE DE DEPART : A$ = ";A$
260 LPRINT "CHAINE TRAITEE : B$ = ";B$
270 LPRINT              ' SAUT DE LIGNE
280 END
1000 ' ROUTINE DE TEST ET DE TRANSFORMATION EVENTUELLE EN MAJUSCULE
1010 '
1020 C$=FNP$(I)         ' Le premier caractère de la chaîne
1030 '                    est mémorisé dans C$
1040 V=ASC(C$)          ' La valeur numérique du caractère C$ en code
1050 '                    ASCII est affectée à V
1060 IF V=>65 AND V=<90 THEN RETURN     ' Le caractère est déjà en majuscule.
1062 '                                    Pas de transformation à faire.
1070 IF V=>97 AND V=<122 THEN C$=FNV$(V)
1080 RETURN             ' SI V n'est pas compris entre 97 et 122, le caractère
1090 '                    n'est pas une lettre et ne peut donc être converti
2000 ' ROUTINE DE TEST ET DE TRANSFORMATION EVENTUELLE EN MINUSCULE
2010 C$=FNP$(I)
2020 V=ASC(C$)
2030 IF V=>97 AND V=<122 THEN RETURN    ' Le caractère est déjà en minuscule.
2040 '                                    Pas de transformation à faire.
2050 IF V=>65 AND V=<90 THEN C$=FNZ$(V)
2060 RETURN
```

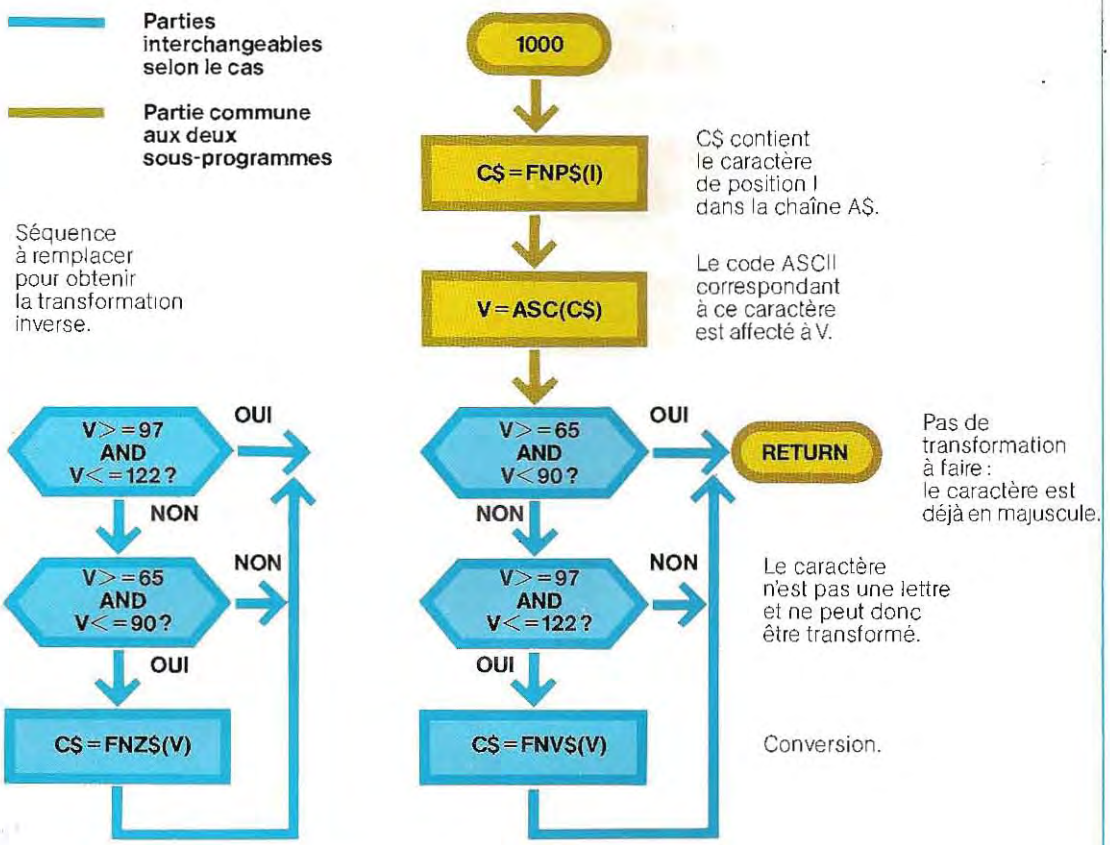
```
CHAINE DE DEPART : A$ = boulevard
CHAINE TRAITEE   : B$ = Boulevard

CHAINE DE DEPART : A$ = HAUSSMANN
CHAINE TRAITEE   : B$ = Haussmann

CHAINE DE DEPART : A$ = 75008
CHAINE TRAITEE   : B$ = 75008

CHAINE DE DEPART : A$ = paris
CHAINE TRAITEE   : B$ = Paris
```

FNZ$(V) = CHR$(V+32)
(passage de majuscule en minuscule).

Le sous-programme de transformation de minuscule en majuscule est schématisé page 488, avec la séquence permettant la transformation inverse. Ci-dessus apparaît le listage complet de cette application.

SOUS-PROGRAMME DE TRANSFORMATION EN MAJUSCULE

Entrée A$ = Chaîne contenant en position I le caractère à transformer.



Domaines d'utilisation. La possibilité, pour l'utilisateur, de définir ses propres fonctions est particulièrement intéressante dans deux secteurs : les applications scientifiques, d'une part, et les manipulations de données nécessitant de nombreuses saisies au clavier, d'autre part. Dans ce dernier cas, on peut créer à l'écran des « masques de saisie », mais ceci réclame une programmation plus élaborée et sera détaillé dans le chapitre concernant la gestion de l'écran.

Nous allons maintenant utiliser les fonctions numériques pour un calcul de surface. La méthode employée, quoique satisfaisante, n'est pas d'une grande rigueur, mais elle illustre bien les applications possibles de l'ordinateur en mathématiques appliquées.

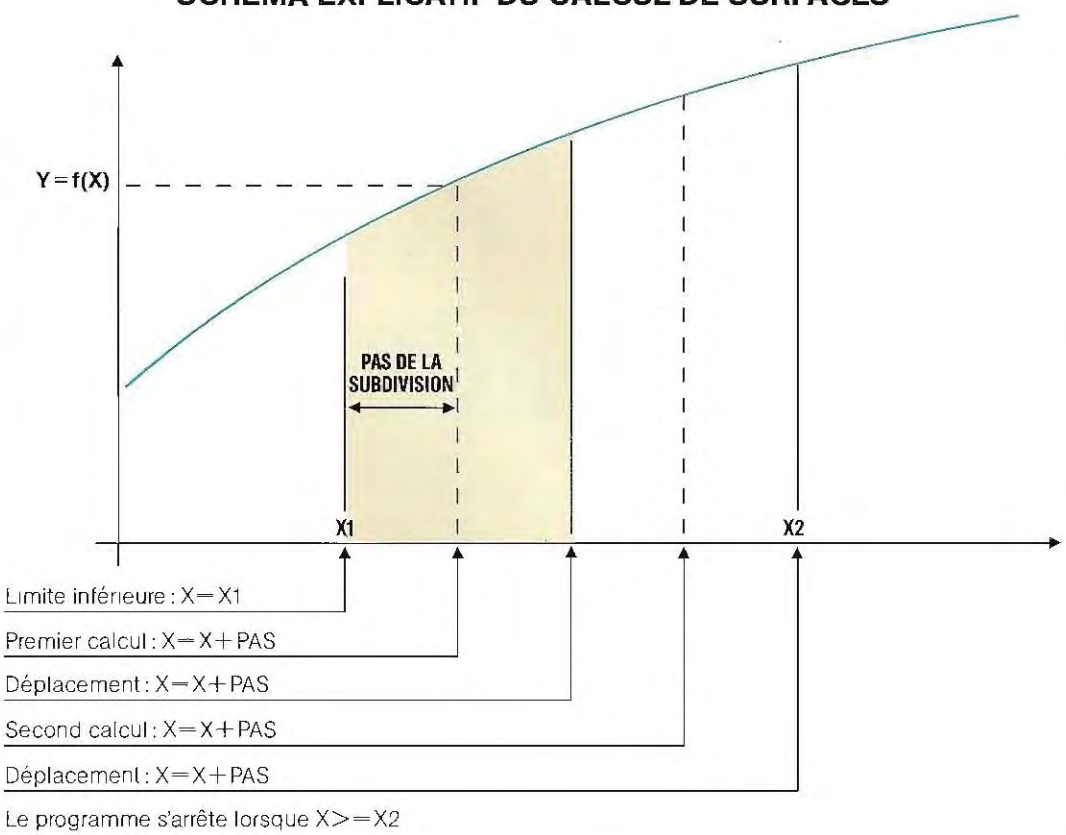
Ce problème a déjà été résolu graphiquement pages 333 et 334. Il s'agissait de décou-

per une surface en petites parties dont on fournissait les coordonnées à l'ordinateur. Celui-ci effectuait alors le calcul de ces surfaces « élémentaires » et les additionnait entre elles. Si l'on connaît l'équation de la courbe (fonction), l'ordinateur peut alors calculer, lui-même et plus précisément, les coordonnées. Il ne reste donc plus qu'à fixer les bornes de l'intervalle et le nombre de subdivisions désiré.

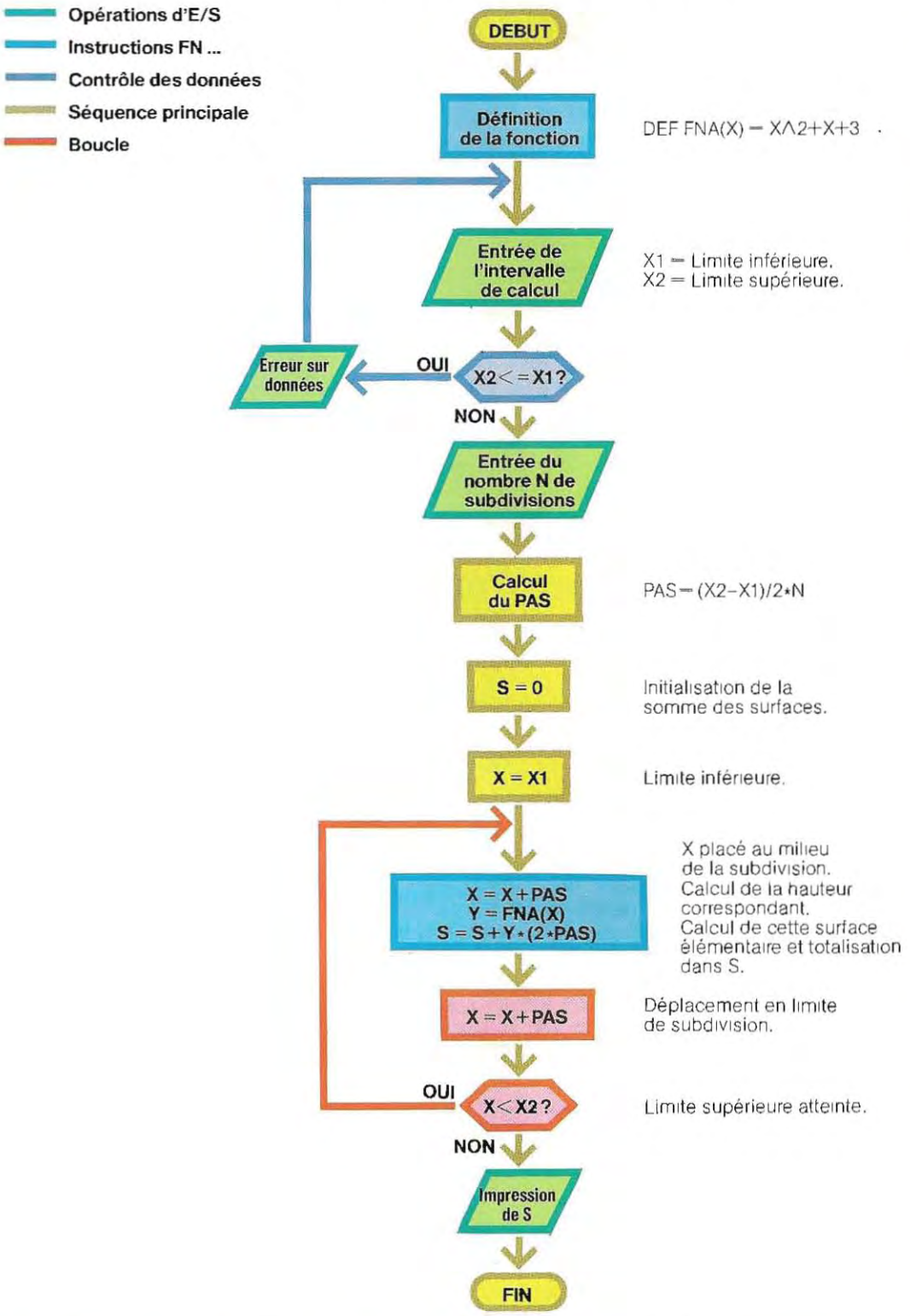
Le listage de la page 491 est accompagné des résultats obtenus pour différents nombres de subdivisions sur l'intervalle [5,25]. La précision du calcul augmente avec ce nombre jusqu'à une limite (80 dans cet exemple) à partir de laquelle l'approximation est suffisante. Il est très difficile de déterminer cette limite a priori. On effectuera donc plusieurs essais afin d'évaluer la précision des résultats et de fixer la valeur qui réalise le meilleur compromis entre fiabilité et durée de traitement.

Dans de tels programmes comportant des

SCHEMA EXPLICATIF DU CALCUL DE SURFACES



ORGANIGRAMME DU CALCUL DE SURFACES



PROGRAMME DE CALCUL DE SURFACES ELEMENTAIRES

```
10 '
20 ' ** PROGRAMME DE CALCUL DE SURFACES ELEMENTAIRES **
30 '
40 '                  PROGRAMME = INTEGRATION
50 '
60 ' FONCTIONS DEFINIES PAR L'UTILISATEUR
70 DEF FNA(X)=X*X*X+X+3          ' Fonction d'une équation du second degré
80 PRINT "ENTREZ LES LIMITES DE LA SURFACE A CALCULER"
90 INPUT "LIMITE INFERIEURE : X1 = ";X1
100 INPUT "LIMITE SUPERIEURE : X2 = ";X2
110 IF X2=<X1 THEN PRINT "DONNEES ERRONEES":GOTO 80
120 PRINT "ENTREZ LE NOMBRE DE SUBDIVISIONS DE LA SURFACE"
130 INPUT "N = ";N
140 '    CALCUL DU PAS
150 '
160 PAS=(X2-X1)/(2*N)
170 S=0          ' MISE A ZERO DE LA SOMME DES SURFACES ELEMENTAIRES
180 X=X1
190 X=X+PAS
200 Y=FNA(X)
210 S=S+Y*(2*PAS)
220 X=X+PAS
230 IF X<X2 GOTO 190
1000 ' INSTRUCTIONS D'IMPRESSION
1002 LPRINT "NOMBRE DE SUBDIVISIONS DE LA SURFACE : ";N
1010 LPRINT "SURFACE TOTALE = ";S
1020 LPRINT
1030 END
```

```
NOMBRE DE SUBDIVISIONS DE LA SURFACE : 10
SURFACE TOTALE = 5520

NOMBRE DE SUBDIVISIONS DE LA SURFACE : 20
SURFACE TOTALE = 5525

NOMBRE DE SUBDIVISIONS DE LA SURFACE : 40
SURFACE TOTALE = 5526.25

NOMBRE DE SUBDIVISIONS DE LA SURFACE : 80
SURFACE TOTALE = 5526.56

NOMBRE DE SUBDIVISIONS DE LA SURFACE : 800
SURFACE TOTALE = 5526.63
```

calculs répétitifs, une version compilée réduit beaucoup le temps d'exécution et autorise un nombre de subdivisions plus important.

Exemple d'application : le chauffage des locaux

L'emploi de sous-programmes s'avère indispensable au-delà d'un certain degré de complexité du programme. Aborder tous les cas possibles et les calculs correspondants au sein d'un seul grand programme mène à un ensemble confus tout en multipliant les risques d'erreur.

L'utilisation de sous-programmes permet, au contraire, d'entrer progressivement dans les détails et d'appréhender plus clairement chaque cas particulier. Après une analyse globale du problème, on dessine l'organigramme principal. Puis, on envisage le développement de tous les sous-programmes annexes.

Le programme idéal comporte ainsi une partie principale, dont le rôle est d'appeler et d'enchaîner une série de sous-programmes spécifiques. Outre sa facilité d'élaboration, un tel programme présente une clarté qui permettra une relecture aisée. Cet aspect est important pour une personne autre que le concepteur, mais aussi pour celui-ci lors de

L'élève et la tortue

L'intérêt des applications éducatives de l'informatique est désormais reconnu aussi bien dans le corps enseignant que dans les milieux professionnels.

L'enseignement fondamental et les exercices pratiques assistés par ordinateur apparaissent dans les centres d'études pédagogiques, les universités et les établissements de formation, mais aussi dans les entreprises, publiques ou privées, et dans les laboratoires de recherche. Ces dix dernières années ont vu se multiplier les logiciels d'enseignement et aujourd'hui, on produit et on diffuse ces systèmes à l'échelle industrielle. De telles applications peuvent revêtir des formes très diverses selon l'orientation choisie.

■ *Certains systèmes permettent de conserver une structure traditionnelle (formation animée par des enseignants à l'école, à l'université ou dans l'entreprise). L'ordinateur n'intervient alors que comme support d'enseignement ou comme complément didactique. Généralement regroupés sous l'étiquette E.A.O. (enseignement assisté par ordinateur), ces systèmes sont l'aboutissement des recherches et des investissements engagés dès 1955 par les universités américaines. L'E.A.O. englobe plusieurs thèmes : enseignement programmé, exercices d'entraînement et de perfectionnement, résolution de problèmes, simulation, jeux éducatifs et contrôle des connaissances.*

■ *D'autres systèmes sont beaucoup plus orientés vers les moyens audiovisuels ou télévisuels et la télématique (liaison avec les banques de données et les conférences). Ils sont surtout conçus pour l'enseignement à distance grâce à des programmes spécifiques diffusés par les réseaux de télévision.*

■ *Enfin les systèmes dits « intelligents », dérivés des recherches menées dans le domaine de l'intelligence artificielle, établissent un dialogue ouvert avec l'élève. Fondés sur la mise au point de programmes simples, ils visent plus au développement de la logique et de la créativité qu'à un apprentissage particulier. Des langages élémentaires mais évolutifs les mettent à la portée des jeunes enfants ou des élèves en difficulté.*

Depuis 1970, la diffusion des ordinateurs touche, dans les pays les plus industrialisés, un nombre toujours croissant d'établissements scolaires. Ces pays mènent, à un niveau national, des politiques de soutien financier au développement de l'informatique pédagogique. Ainsi, aux Etats-Unis, le gouvernement fédéral avait investi, principalement par le biais de la National Science Foundation, près de 300 millions de dollars. A la fin des années 1960, ce budget fut confié au Ministère de l'Education.

C'est à l'instigation du Centre pour la Recherche et l'Innovation dans l'Enseignement (CERI) de l'O.C.D.E. que des pays comme l'Allemagne, la Grande-Bretagne, le Japon ou la France multiplient les expériences depuis 1970. A l'échelle de la C.E.E., c'est le Fonds Social Européen qui finance les projets pédagogiques orientés essentiellement vers la formation professionnelle.

En France, des associations comme l'ADEMIR, regroupées dans une Fédération Nationale (Microtel) et soutenues par le Ministère de l'Education et l'Agence de l'Informatique, contribuent au développement de l'informatique scolaire en créant des « bibliothèques » de didacticiels (logiciels d'enseignement).

De façon générale, il faut souligner que l'intérêt pour l'enseignement assisté par ordinateur s'accompagne aujourd'hui d'exigences plus profondes de la part d'un public qui avait pu mettre en doute les possibilités des « machines à enseigner », à l'époque où elles n'étaient qu'à l'état de projet.

Lorsqu'au début des années 1960, l'ordinateur fit son apparition dans les écoles et les universités américaines, le milieu enseignant était très attiré par des théories pédagogiques comme celle de l'enseignement programmé. La première génération des systèmes d'E.A.O. naquit, dans ce contexte, des recherches menées par Patrick Suppes et Richard Atkinson à l'Institut de mathématiques appliquées aux sciences sociales de l'Université de Stanford, par Donald Bitzer à l'Université de l'Illinois (système Plato) et par les laboratoires d'IBM (système CAI 1500). Les expériences menées dans les années 1960 furent fortement influencées par la technologie caractéristique de cette période.

Les stratégies fondées sur la notion d'enseignement programmé impliquaient un contrôle rigide du dialogue avec l'ordinateur. Toutefois, dès 1965, on vit se développer une

nouvelle approche théorique et expérimentale de l'utilisation de l'ordinateur dans l'enseignement. A la conception de l'ordinateur, simple machine à « tourner les pages » se bornant à automatiser l'enseignement, succéda une nouvelle conception dont la devise aurait pu être, selon les mots de Seymour Papert : « arrangeons-nous pour que ce soit les enfants qui programment l'ordinateur et non l'ordinateur qui programme les enfants ».
Après plusieurs années de collaboration avec le psychologue suisse Jean Piaget, Papert, spécialiste en cybernétique, assura, aux côtés de Marvin Minsky, la direction du Laboratoire d'Intelligence Artificielle du MIT. Selon lui, l'enfant n'apprend rien de l'ordinateur sinon en le programmant, ce qui l'oblige à exprimer clairement ses « intuitions ». Traduire une idée sous la forme d'un programme, c'est la concrétiser et en faire un objet de réflexion.
Cette approche, telle qu'elle ressort de la biographie de Papert, s'appuie sur les concepts de la science cognitive en matière d'intelligence artificielle, et elle a évolué dans un environnement expérimetal en même temps que les langages interactifs.
Historiquement, le Basic est considéré comme le premier langage ayant permis un enseignement basé sur la programmation. Toutefois, celui-ci se limite à des calculs numériques, du moins dans sa version classique. Des langages interactifs plus récents donnent, par contre, accès à des domaines plus variés. Ainsi, le LOGO et le SMALLTALK offrent à l'enfant des possibilités de rédaction, de graphisme ou de symbolisation.
Tout langage dispose d'un certain nombre d'instructions de base. On réalise alors un programme en introduisant une suite logique utilisant ces commandes.
Avec un langage comme le LOGO, il est possible de définir de nouvelles instructions qui pourront être utilisées dans des programmes ultérieurs.
Une instruction, correspondant à une opération complexe, peut ainsi, au terme d'une analyse attentive, être déterminée comme une suite d'opérations plus simples. Cette manière d'aborder un problème par paliers successifs de complexité décroissante est la base de toute découverte scientifique.
Connaître, c'est traduire l'inconnu par le biais d'un raisonnement logique dont on maîtrise chaque étape. Cette méthode scientifique est aussi bien adaptée à l'enseignement des matières les plus variées et ce, quel que soit l'âge de l'élève. De plus, savoir décomposer une structure complexe est une capacité indispensable à la conception et à l'accomplissement de projets à long terme.
Le SMALLTALK, pour sa part, fut mis au point par Alan Kay au Centre de recherches Xerox de Palo Alto. Tout d'abord destiné à faciliter les conditions de programmation, il fut ensuite développé à des fins didactiques et s'avère aujourd'hui très adapté à la création de dessins animés, de jeux et de compositions musicales.
A l'origine du SMALLTALK, se trouve une assimilation de toute démarche intellectuelle à la coopération et la compétition de plusieurs processus mentaux. Si l'on désire, par exemple, dessiner une figure comportant deux types de symétrie, on peut imaginer une procédure pour créer la première symétrie, puis l'essayer pendant qu'on réfléchit à la seconde. Une fois la seconde procédure écrite, on reprendra la première pour juger des résultats obtenus tandis que se déroulera la seconde. Quand tout est mis au point, l'exécution simultanée des deux procédures fournit le résultat final, c'est-à-dire le dessin désiré. Il est évident que l'utilisation de cette méthode parallèle facilite grandement la programmation de fonctions complexes.
L'emploi des langages interactifs a été expérimenté – et ce, principalement avec des enfants – par Papert et Minsky. Ils s'inspiraient des conceptions de Piaget selon lesquelles l'intelligence est une adaptation à l'environnement et l'éducation une évolution spontanée à travers différents stades intellectuels.
De cette théorie découle l'idée qu'il est nécessaire de mettre le sujet (généralement, un enfant) dans des conditions propices à toute étude. C'est ainsi que furent créés le Laboratoire d'Intelligence Artificielle du MIT et son atelier d'enseignement assisté par ordinateur où fut développé ce langage de programmation très simple qu'est le LOGO.
Toutes sortes de gens ont pu se mesurer à ce langage : des enfants de quatre ans, des classes entières d'élèves venant de régions des environs de Boston culturellement désavantagées, des lycéens surdoués, des étu-

diants en sciences habitués aux ordinateurs du MIT, des chercheurs d'universités latino-américaines (où le LOGO fut importé) et des ingénieurs d'autres laboratoires d'Intelligence Artificielle comme celui d'Edimbourg. Enfin, les chercheurs les plus brillants du laboratoire ont testé le LOGO et sa conception de l'enseignement. Les résultats de ces travaux firent l'objet de plus de deux cents rapports techniques décrivant les réalisations les plus disparates, de l'écriture automatique de poèmes à la simulation de phénomènes biologiques ou comportementaux.

Le LOGO permet de dialoguer avec l'ordinateur et de lui apprendre de plus en plus de choses en fonction de ce qu'il sait déjà.

L'ordinateur devient alors un véritable partenaire qui exécute fidèlement les ordres, à condition qu'ils soient exprimés de façon correcte. Le LOGO ne reconnaît au départ qu'un nombre limité de mots qui permettent de définir d'autres commandes. C'est un robot capable d'apprendre : on peut étendre son langage en associant, à l'aide des instructions de base, des opérations à des mots nouveaux qui seront alors ajoutés à son vocabulaire de base. Orientée au départ vers les matières traditionnelles, l'utilisation du LOGO a rapidement dépassé le cadre de ces disciplines pour parvenir à des expériences plus ouvertes exploitant toutes les possibilités de l'ordinateur.

La plus célèbre de ces expériences, également réalisée avec d'autres langages comme le SMALLTALK ou le LISP, langage de base de l'intelligence artificielle, est celle de la « Tortue ». La Tortue est un véritable petit automate téléguidé par l'intermédiaire du langage LOGO. La Tortue avance en laissant derrière elle une trace qui, au fur et à mesure de ses déplacements, réalise un dessin sur l'écran.

La Tortue est généralement représentée par un point ou un petit triangle lumineux. On peut tracer n'importe quelle figure grâce aux commandes de base qui sont des ordres très simples : AVANCE, RECULE, DROITE, GAUCHE, indicés par la longueur du trajet ou par l'angle en degrés de la rotation à effectuer. Ces commandes simples peuvent être agencées en séquences plus ou moins complexes qui finissent par constituer de véritables procédures. L'utilisateur les imagine et les crée de façon autonome, faisant appel à son intuition et à la logique, au gré d'essais plus ou moins fructueux.

Les instructions AVANCE et RECULE provoquent le déplacement de la Tortue le long de l'axe sur lequel elle se trouve.

Pour modifier cette orientation, on doit employer les commandes DROITE et GAUCHE qui font alors pivoter la Tortue sur elle-même.

Ces quatre commandes s'accompagnent d'un indice correspondant soit à la longueur du déplacement, soit à l'angle de rotation. Déjà, dans ce dernier cas, l'expression d'un angle en degrés, si elle semble évidente à des adultes, constitue une véritable découverte pour les enfants.

Ainsi, la séquence d'instructions suivante apprend à l'ordinateur une nouvelle commande (ou mot) appelée « CARRÉ » :

```
POUR CARRÉ
AVANCE 100
DROITE 90
AVANCE 100
DROITE 90
AVANCE 100
DROITE 90
AVANCE 100
DROITE 90
FIN
```

ou encore :

```
POUR CARRÉ
RÉPÈTE 4
AVANCE 100
DROITE 90
FIN
```

soit plus généralement :

```
POUR CARRÉ : N
RÉPÈTE 4
AVANCE : N
DROITE 90
FIN
```

On pourra définir le tracé d'un triangle équilatéral d'une façon analogue :

```
POUR TRIANGLE
AVANCE 100
DROITE 120
AVANCE 100
DROITE 120
AVANCE 100
DROITE 120
FIN
```

ou encore :

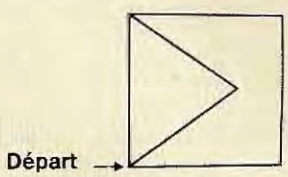
```
POUR TRIANGLE : N
```

RÉPÈTE 3
AVANCE : N
DROITE 120
FIN

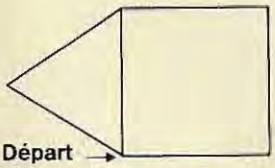
On voit bien que l'intérêt de ce langage réside moins dans le dessin en soi que dans la construction même du programme, élément par élément. Ce travail permet à l'enfant de hiérarchiser ses connaissances et de développer peu à peu une souplesse et une vivacité intellectuelle, qu'il aura ensuite très souvent l'occasion de mettre à profit. Cette façon d'appréhender les problèmes transparaît incontestablement dans les projets entrepris par les enfants après quelques séances de travail avec la Tortue.
La définition du «mot» MAISON est un exemple très simple mais cependant significatif. On devine facilement qu'après avoir défini les mots CARRÉ et TRIANGLE, il suffira pour dessiner une maison de disposer un triangle au-dessus d'un carré. Voici une première ébauche du programme :

POUR MAISON
CARRÉ
TRIANGLE
FIN

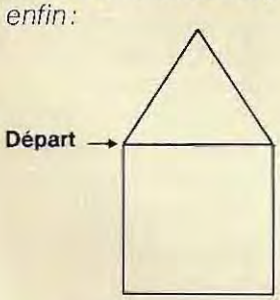
L'exécution de ce programme donne ce résultat...



Le triangle n'a pas été dessiné au-dessus mais à l'intérieur du carré !
Pour comprendre d'où provient l'erreur, il suffit de suivre le trajet parcouru par la Tortue ; on se rendra alors compte que le triangle est mal positionné parce que les deux mots sont définis de manière analogue et que le triangle vient donc se superposer sur le carré. On peut donc résoudre ce problème soit en redéfinissant le TRIANGLE avec des rotations de 120° vers la gauche (et non vers la droite), soit en modifiant l'orientation de la Tortue entre les deux figures.
En insérant l'instruction GAUCHE 60 entre CARRÉ et TRIANGLE, on obtient ceci :



L'apprenti-programmeur fait des progrès. Il réalise que les choses ne sont pas forcément justes ou fausses et qu'il peut y avoir une continuité d'un cas à l'autre. Cette deuxième maison a plus d'allure que la précédente mais ne sera parfaite qu'une fois redressée. Pour cela, il convient de faire pivoter la Tortue d'un quart de tour avant le début du dessin. Avec l'instruction DROITE 90 au début du programme, on obtient enfin :



Outre la Tortue et ses dessins, le LOGO permet : 1) la rédaction de poèmes fondée sur un remaniement créatif de la grammaire et sur la génération aléatoire de mots d'une catégorie déterminée ; 2) l'étude empirique de la régularité des nombres et des opérations arithmétiques ; 3) des expériences de composition musicale avec une « boîte à musique » commandée par un ordinateur. Le LOGO autorise une décomposition de la musique en éléments fondamentaux (mélodie, rythme, harmonie et couleur). En associant ces différents paramètres à l'aide d'instructions appropriées, on réussit à en saisir l'importance et la valeur réelle. Dans cette optique cognitive et interactive qui est à l'origine de langages comme le LOGO ou le SMALLTALK, l'ordinateur devient un instrument dont la programmation permet à chacun de formaliser ses connaissances et d'acquérir des capacités d'analyse et de résolution de problèmes complexes. Cette conception va donc bien plus loin que la notion traditionnelle d'E.A.O. et joue en faveur de l'introduction de l'informatique dans le milieu scolaire, et ce, bien avant toute orientation professionnelle.

(L'exemple est extrait de « Mindstorms », de S. Papert.)

la mise au point ou d'interventions ultérieures. Les étapes préliminaires peuvent paraître fastidieuses et sans grand rapport avec l'utilisation de la machine mais il faut être conscient du fait qu'un ordinateur ne peut traiter que des problèmes dont il connaît toutes les composantes. Un logiciel rédigé à la hâte est, dans le meilleur des cas, peu pratique, et, au pire, incomplet.
Il est possible d'informatiser les tâches les plus variées. Aussi le programmeur se trouve-t-il souvent confronté à des problèmes et à des domaines qu'il ne connaît pas.
L'utilisateur doit alors prendre part à la préparation du logiciel en fournissant au programmeur toutes les informations et les méthodes de calcul dont il pourra avoir besoin pour l'application en question.
Pour illustrer l'analyse d'un problème relativement complexe, nous allons suivre l'élaboration d'un programme calculant les déperditions calorifiques d'un local.
Le travail se déroule en six étapes :

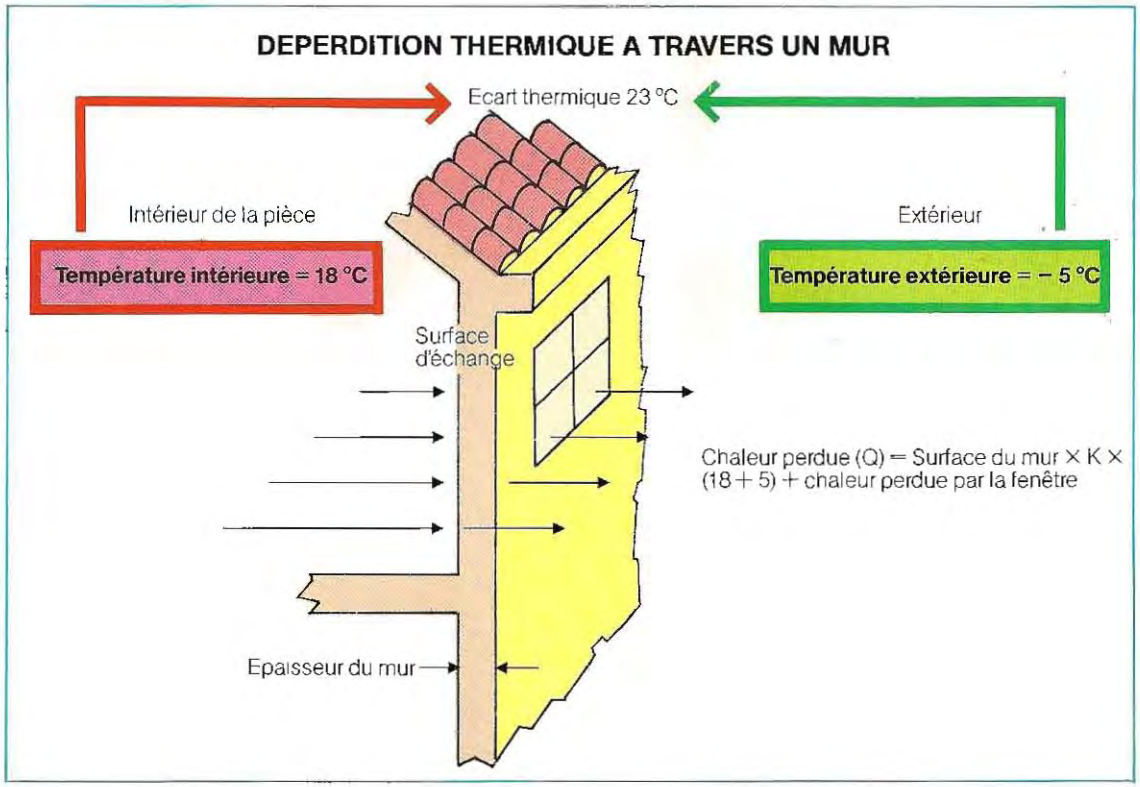
1 / une analyse générale, où sont définies les différentes variables et méthodes de traitement ;
2 / la synthèse, puis la préparation de l'agencement du programme principal ;
3 / une analyse détaillée et l'élaboration des organigrammes de chaque sous-programme ;
4 / l'écriture de ces sous-programmes, puis leur mise au point ;
5 / l'écriture du programme principal ;
6 / et enfin, la réunion du programme principal et des sous-programmes, assortie des derniers tests.

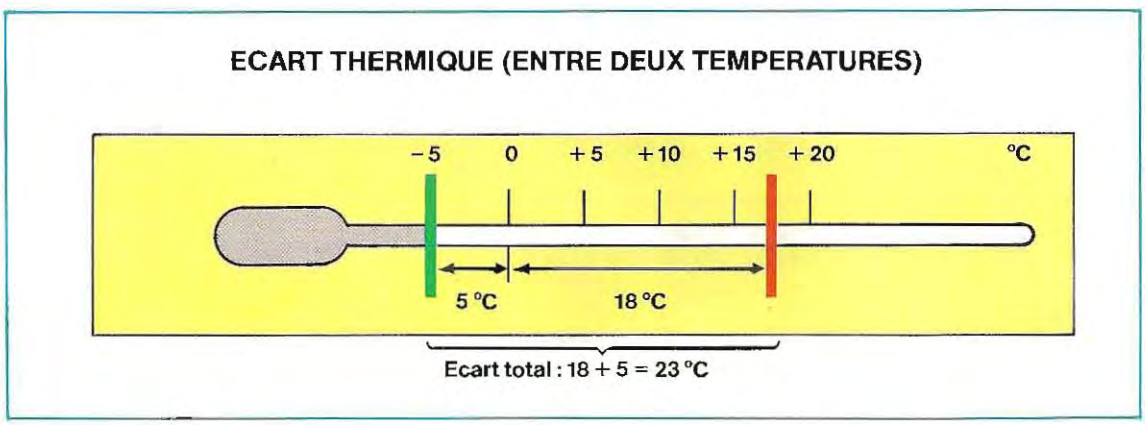
On voit que chaque sous-programme est réalisé et mis au point séparément avant d'être intégré à l'ensemble qui est testé à son tour. Nous verrons plus loin d'autres méthodes de contrôle des logiciels.

Analyse générale du problème. Il faut, tout d'abord, bien cerner le problème à résoudre et considérer l'ensemble des facteurs intervenant. Le programmeur travaillant pour un client a alors besoin de sa collaboration pour bien définir les objectifs à atteindre et les calculs à effectuer.
Dans notre exemple, il est indispensable de connaître les lois physiques régissant les échanges de chaleur.
On peut assimiler une pièce à un volume dont les parois séparent l'air intérieur de l'air extérieur. Les températures de l'air de part et d'autre des murs tendent à s'équilibrer par des déperditions de chaleur plus ou moins rapides selon la nature des parois. La pièce se refroidit donc en permanence et seul un apport de chaleur équivalent permet de maintenir une température constante.
Ainsi, pour une pièce perdant 1250 calories par heure, il est facile de déduire la quantité de mazout (par exemple) nécessaire au maintien de la température.
Ce calcul de consommation peut s'effectuer pour n'importe quelle source d'énergie et le but du programme est de déterminer la déperdition à compenser. Celle-ci dépend de nombreux facteurs et sera évaluée pour chaque pièce du local considéré. Il suffira ensuite d'additionner l'ensemble des pertes calculées pour connaître (en calories par heure) la puissance nécessaire au chauffage de tout le bâtiment. La quantité de chaleur rayonnant vers l'extérieur à travers une paroi est donnée par la relation suivante (voir tableau page 498) :

$$Q = \left[\begin{matrix}\text{Surface} \\ \text{du mur}\end{matrix}\right] \times K \times \left[\begin{matrix}\text{Température} \\ \text{intérieure}\end{matrix} - \begin{matrix}\text{Température} \\ \text{extérieure}\end{matrix}\right] \quad (1)$$

Q est la quantité de chaleur perdue exprimée en calories/heure et K est un coefficient de transmission dépendant du matériau et de l'épaisseur de la paroi.
Ce coefficient représente la capacité d'isolation d'une séparation.
Il est bien évident que les déperditions seront supérieures à travers une cloison qu'à travers un mur, mais moindres que par une vitre.
L'expression [Température intérieure – Température extérieure] représente l'écart de température de part et d'autre de la séparation. Plus grande sera cette différence et plus la pièce aura tendance à se refroidir vite. (Attention au calcul de l'écart thermique pour des températures extérieures inférieures à zéro : voir schéma page 497).
La déperdition est enfin proportionnelle à la surface d'échange entre les deux milieux. Les valeurs de K, indispensables au calcul, sont répertoriées dans des tables spécialisées





concernant les différents matériaux utilisés. Plus nous fournirons de valeurs de K à l'ordinateur, plus le programme sera « universel ». Nous n'utiliserons dans l'exemple qu'une table très rudimentaire, mais la conception du programme permet d'ajouter autant de données que l'on veut. Les valeurs actuellement retenues figurent dans la table de la page 498.

Avant d'aborder la conception des organigrammes, nous allons nous familiariser avec la méthode de calcul en résolvant « à la main » l'un des cas qui peuvent se présenter.

Type de paroi	Epaisseur (en cm) 25	38	50	Symbole	Table entrée dans l'ordinateur
Mur en briques	1.6	1.3	1.0	MB	KB(1), KB(2), KB(3)
Mur en ciment	2.2	1.8	1.6	MC	KC(1), KC(2), KC(3)
Sol		1.0		SL	
Toit		2.0		TO	
Porte		4.0		PO	
Fenêtre		5.0		FE	

KB(1) = 1.6
KB(2) = 1.3
KB(3) = 1.0

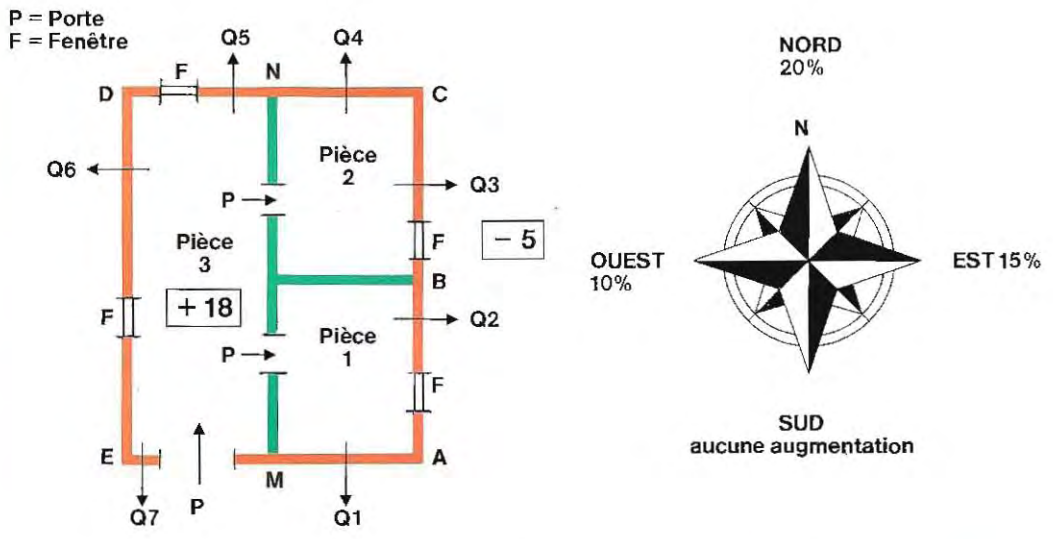
KC(1) = 2.2
KC(2) = 1.8
KC(3) = 1.6

Cette table donne les valeurs de K pour un certain nombre de parois dont le programme reconnaît les symboles indiqués dans l'avant-dernière colonne.

On a choisi une épaisseur moyenne pour les derniers éléments (sol, etc.) car ce paramètre n'influe que très peu sur la valeur de K correspondante.
Le programme est constitué d'une suite d'instructions paramétrées qui feront effectuer les mêmes opérations à la machine. Prenons le cas de l'appartement dont le plan figure ci-dessous.
Les traits verts représentent les murs intérieurs. Aucun échange thermique ne s'y produit puisque les différentes pièces sont à la même température (écart thermique nul). La déperdition de chaleur se fait donc à travers les murs extérieurs (en rouge), le toit et le sol. Le calcul sera effectué pièce par pièce. La maison présente trois salles dont les pertes sont réparties entre :

- les murs externes ;
- les fenêtres et les portes donnant sur l'extérieur ;

PLAN DU TROIS PIECES PRIS EN EXEMPLE



- le sol et le plafond ;
- le renouvellement de l'air (aération).

Ainsi, dans la pièce 1, des déperditions sont occasionnées par :

- les murs : Q2 + Q1 (Q2 sur le côté AB et Q1 sur le côté MA) ;
- une fenêtre ;
- le sol et le plafond : pertes proportionnelles à la surface de la pièce (côté AB × côté MA).

On doit également tenir compte de l'exposition des murs. Ainsi, un mur orienté au nord verra sa perte de chaleur majorée de 20 %. Les corrections respectives pour les murs est et ouest sont de 15 % et de 10 %.
Seule l'exposition au sud n'entraîne aucune perte supplémentaire. Le mur AB de la pièce 1 subira une déperdition de 15 % supérieure à celle tirée de la formule (1).
Supposons, par exemple, que le mur AB mesure 5 m, le mur AM 4 et qu'ils soient tous les deux construits en briques de 38 cm d'épaisseur sur une hauteur de 3,50 m ;
K = 1.3 (voir table page 498).
Pour le mur est (AB) nous aurons donc :

$$\text{Surface} = 5 \times 3{,}50 = 17{,}50 \text{ m}^2$$

dont il faut déduire la surface de la fenêtre (2 m^2) car sa contribution aux pertes sera calculée séparément (K différent de celui du mur).
La surface d'échange est donc :

$$\text{Surface} = 17{,}50 - 2 = 15{,}50 \text{ m}^2$$

Après application de la formule (1) :

$$\text{Chaleur perdue (Q1)} = 15{,}50 \times 1{,}3 \times (18 + 5)$$

on trouve :

$$Q1 = 463{,}45 \text{ kcal/heure}$$

Nous devons encore majorer cette valeur de 15 % en raison de l'exposition du mur :

$$Q1 = 533 \text{ kcal/heure (valeur arrondie)}$$

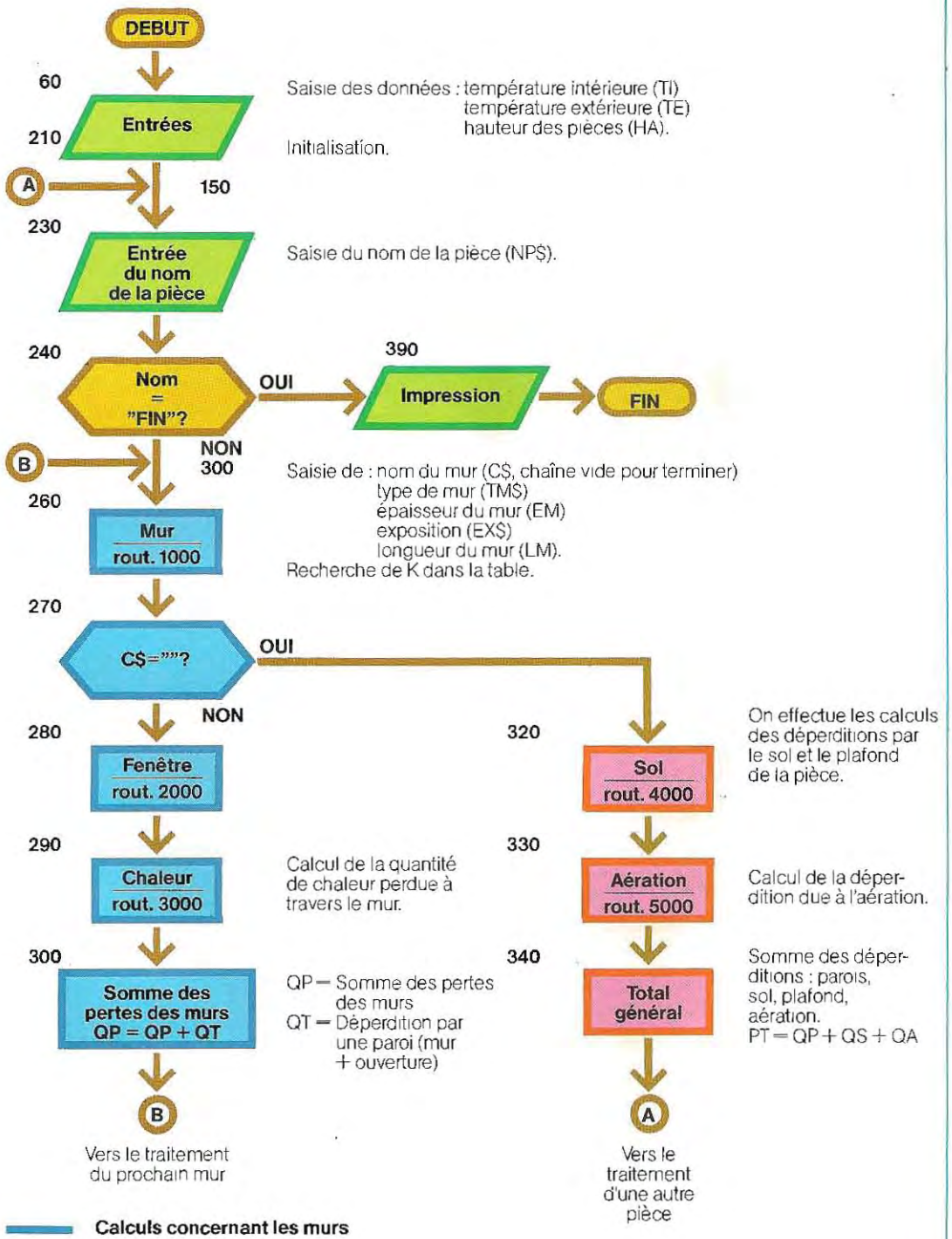
A elle seule, la fenêtre (surface 2 m^2, K = 5) émet 2 × 5 × (18 + 5) soit 230 kcal/heure, et donc, après correction (15 % pour l'est), 265 kcal/heure. La quantité de chaleur rayonnée par la façade AB s'élève ainsi à :
533 + 265 = 798 kcal/heure.
On pourrait effectuer le même calcul pour le mur MA, puis pour le toit (K = 2, surface = côté AB × côté MA) et enfin pour le sol (K = 1, même surface).
Toutefois, la somme de ces résultats ne représente pas la totalité des déperditions de la pièce 1, car il faut aussi tenir compte de la chaleur perdue avec le renouvellement de l'air.
L'explusion d'air chaud et son remplacement par de l'air froid provoque une perte que l'on peut évaluer ainsi :

$$\text{Renouvellement} = \left[\begin{matrix}\text{Volume de} \\ \text{la pièce}\end{matrix}\right] \times 0.3 \times \left[\begin{matrix}\text{Ecart de} \\ \text{température}\end{matrix}\right]$$

La quantité de chaleur nécessaire pour maintenir la pièce à une température constante est donc égale à la somme des calories émises à travers les parois et de celles véhiculées par les courants d'air.
En réitérant ce calcul pour l'ensemble des pièces, on arrive à déterminer la puissance requise pour l'installation de chauffage.
Nous avons maintenant terminé l'analyse générale et on doit en faire la synthèse avant d'élaborer l'organigramme principal.
Récapitulons les étapes de cette procédure :

1 / saisie des constantes : températures intérieure et extérieure, hauteur des pièces ;
2 / puis, pour chaque pièce, saisie du nom de la pièce (ou du mot FIN quand toutes les pièces ont été traitées) ;
3 / pour chaque mur externe, saisie du matériau, de la longueur et de l'exposition ;
4 / saisie de la surface des portes et des fenêtres éventuelles ;
5 / calcul des pertes de chaleur à travers ce mur ;
6 / retour à l'étape 3 pour les autres parois externes de cette pièce ;
7 / saisie de la surface du sol et du toit ;
8 / calcul des déperditions à travers le sol et le toit ;
9 / calcul de la chaleur perdue lors du renouvellement d'air (aération) ;
10 / addition de toutes les pertes de chaleur de cette pièce ;
11 / retour à l'étape 2 pour une autre pièce ;
12 / calcul et impression du total général.

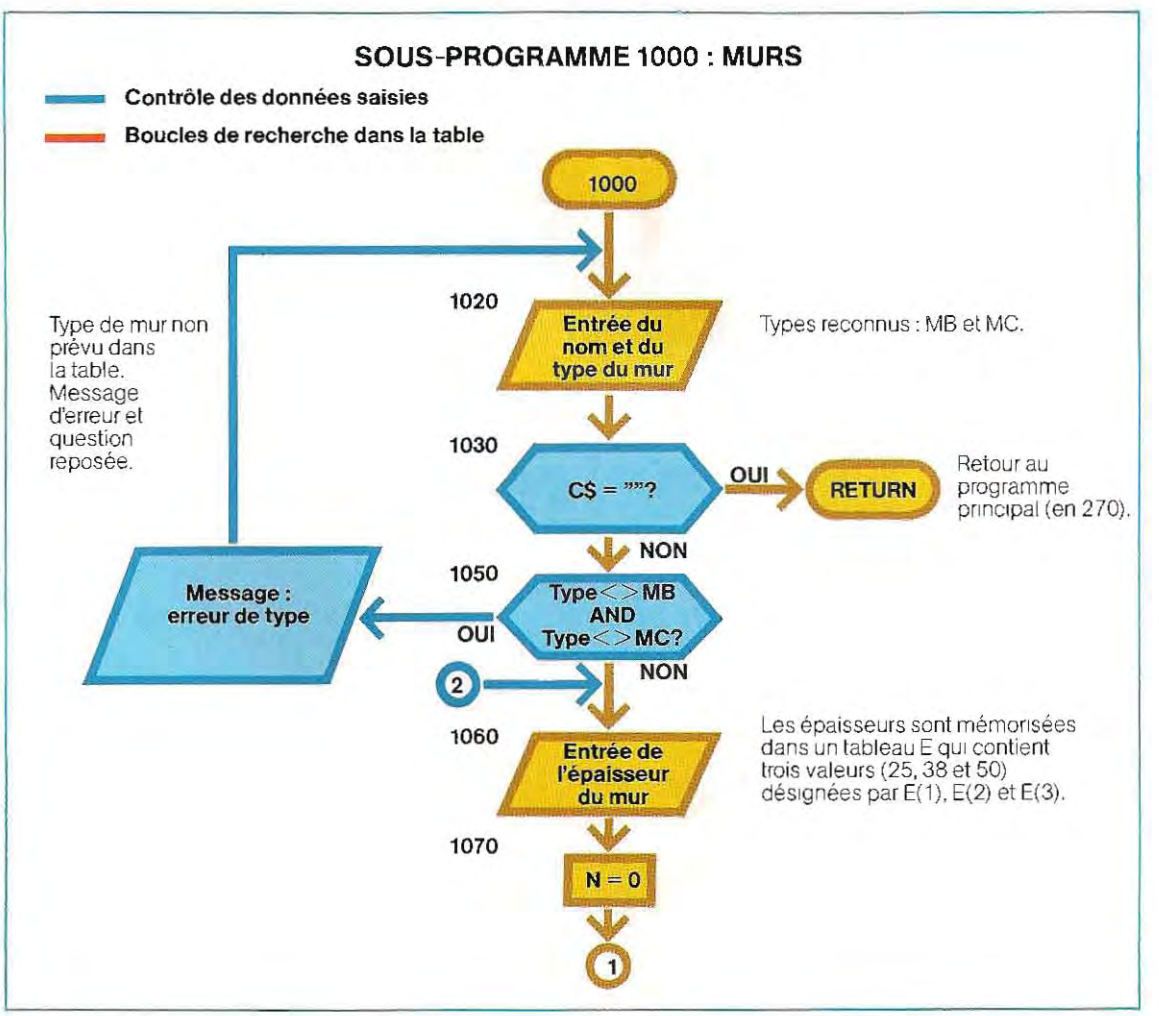
CALCUL DES DEPERDITIONS THERMIQUES
ORGANIGRAMME DU PROGRAMME PRINCIPAL

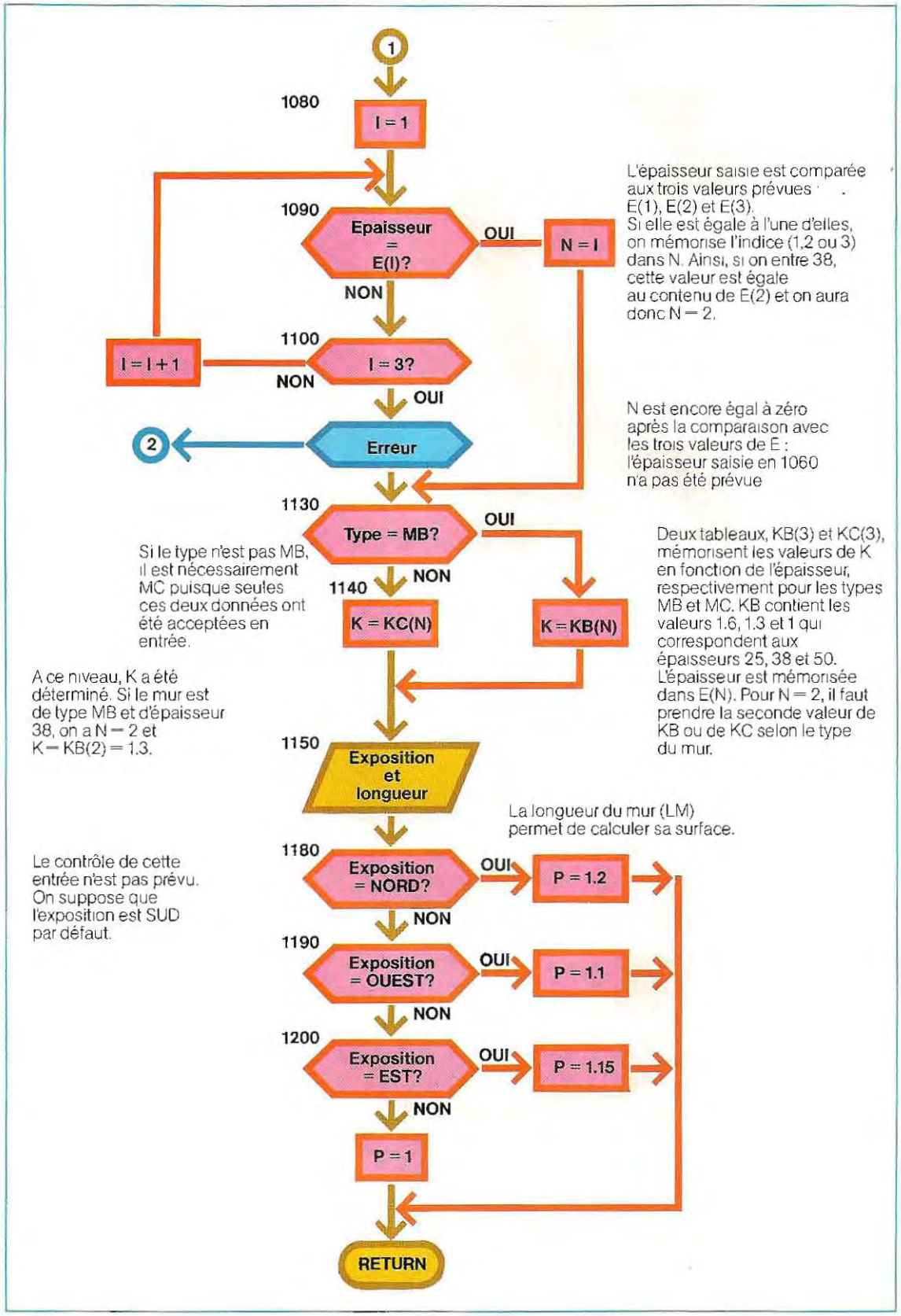


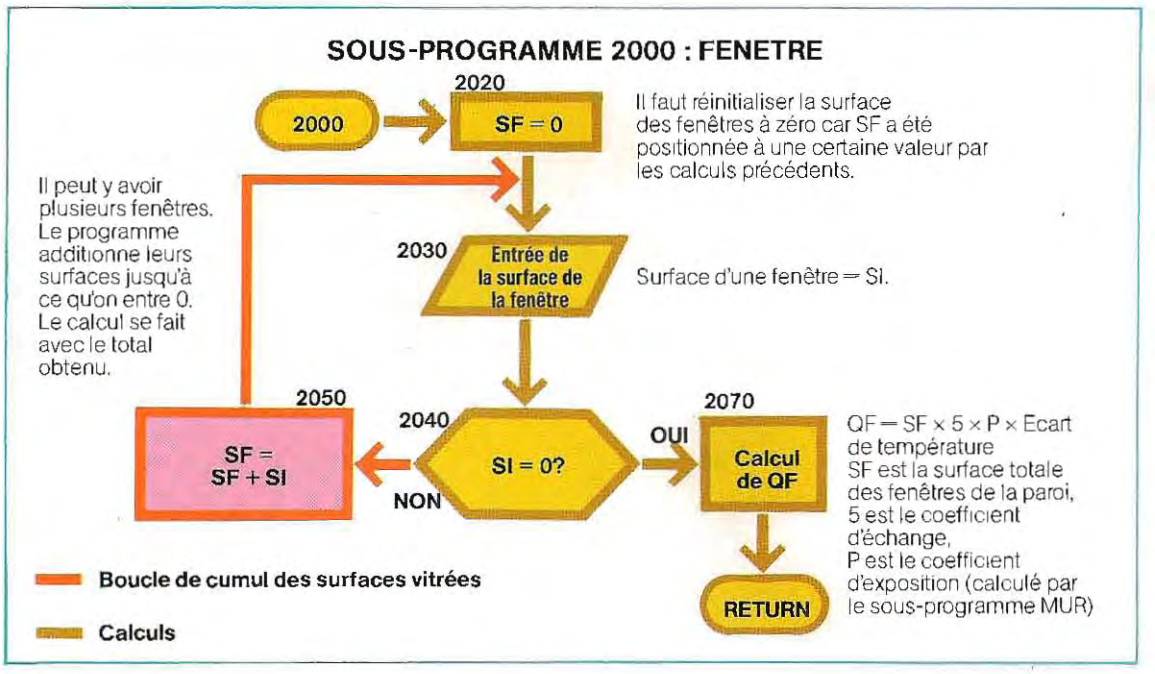
L'organigramme principal présenté page 500 montre l'enchaînement logique des opérations à effectuer. Il faudra traduire ces opérations en Basic pour obtenir le programme proprement dit. Les blocs représentant des calculs seront traités par des sous-programmes (routines) débutant chacun à la ligne indiquée (par exemple, la déperdition par une fenêtre sera calculée par la routine 2000).

Les sous-programmes. Avant d'élaborer l'organigramme des différents sous-programmes, on doit approfondir l'analyse en définissant avec précision les valeurs utilisées par chacun d'eux, les calculs à effectuer et les résultats à dégager.
Voici le détail des différents traitements.
Sous-programme 1000 : MURS
Déroulement :
Saisie des caractéristiques du mur.
Vérification de la concordance de la donnée avec une valeur de la table (symbole de deux lettres, voir page 498).
Lecture du coefficient K correspondant.
Détermination de la correction due a l'exposition.
Sorties :
K = coefficient d'échanges thermiques.
P = coefficient d'exposition (P = 1.2 pour le nord, 1.15 pour l'est, 1.1 pour l'ouest et 1 pour le sud).
LM = longueur du mur.
Tableaux utilisés :
E(3) : épaisseurs de mur possibles.
KB(3) : coefficients thermiques de la brique.
KC(3) : coefficients thermiques du ciment.

Sous-programmes 2000 : FENETRE
Déroulement :
Boucle de saisie de la surface des ouvertures



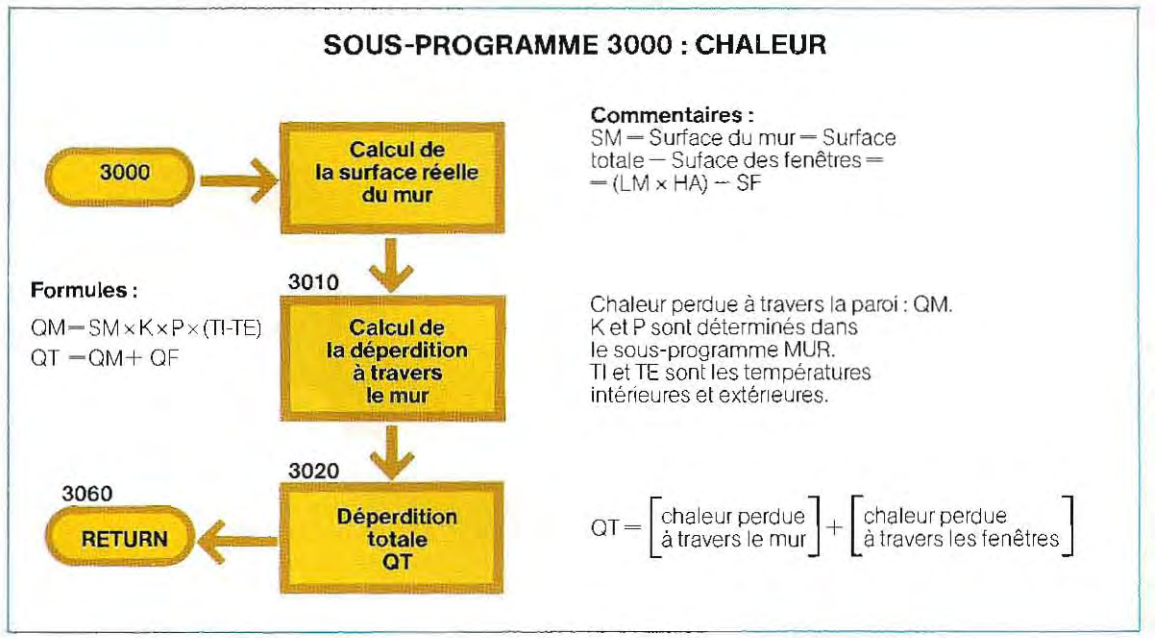

1
1080
I = 1
1090
Epaisseur = E(I)?
OUI
N = I
NON
1100
I = I + 1
I = 3?
NON
OUI
2
Erreur
1130
Type = MB?
OUI
NON
1140
K = KC(N)
K = KB(N)
1150
Exposition et longueur
1180
Exposition = NORD?
OUI
P = 1.2
NON
1190
Exposition = OUEST?
OUI
P = 1.1
NON
1200
Exposition = EST?
OUI
P = 1.15
NON
P = 1
RETURN
L'épaisseur saisie est comparée aux trois valeurs prévues E(1), E(2) et E(3). Si elle est égale à l'une d'elles, on mémorise l'indice (1,2 ou 3) dans N. Ainsi, si on entre 38, cette valeur est égale au contenu de E(2) et on aura donc N = 2.
N est encore égal à zéro après la comparaison avec les trois valeurs de E : l'épaisseur saisie en 1060 n'a pas été prévue
Deux tableaux, KB(3) et KC(3), mémorisent les valeurs de K en fonction de l'épaisseur, respectivement pour les types MB et MC. KB contient les valeurs 1.6, 1.3 et 1 qui correspondent aux épaisseurs 25, 38 et 50. L'épaisseur est mémorisée dans E(N). Pour N = 2, il faut prendre la seconde valeur de KB ou de KC selon le type du mur.
Si le type n'est pas MB, il est nécessairement MC puisque seules ces deux données ont été acceptées en entrée.
A ce niveau, K a été déterminé. Si le mur est de type MB et d'épaisseur 38, on a N = 2 et K = KB(2) = 1.3.
La longueur du mur (LM) permet de calculer sa surface.
Le contrôle de cette entrée n'est pas prévu. On suppose que l'exposition est SUD par défaut.



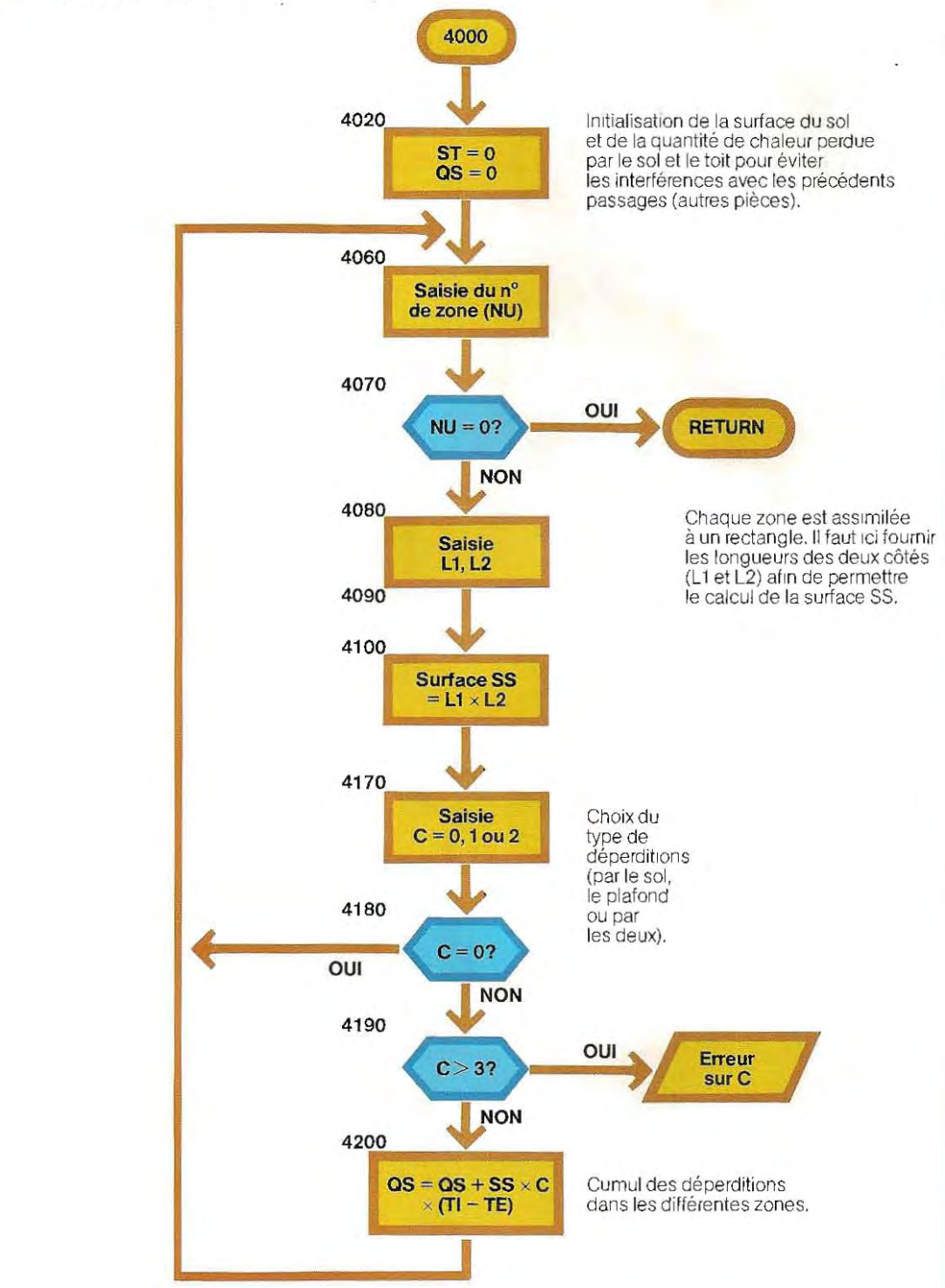
quel que soit leur nombre.
Calcul de la quantité de chaleur qui s'échappe par les fenêtres.
Sorties :
SF = Surface des fenêtres (à soustraire de la surface des murs).
QF = Pertes de chaleur à travers les fenêtres.

Sous-programme 3000 : CHALEUR
Déroulement :
Calcul de la quantité de chaleur perdue par le mur et de la déperdition totale (mur + fenêtres éventuelles).
Sorties :
QM = Déperdition totale de cette façade.

Sous-programme 4000 : SOL
Déroulement :
Calcul des pertes de chaleur par le sol et par le toit.



SOUS-PROGRAMME 4000 : SOL



Sorties :
QS = Total des déperditions de chaleur par le sol et par le plafond.
Il faut effectivement préciser si de la chaleur se dissipe par le sol ou par le plafond : si la pièce est située au-dessus (ou en-dessous) d'une autre pièce chauffée, il n'y a pas de déperditions par le sol (ou par le plafond).

Sous-programme 5000 : AERATION

Opérations exécutées :
Calcul de la chaleur perdue pendant l'aération.

Sorties :
QA = Chaleur perdue pendant l'aération.

Le programme. Le listing complet et commenté du programme de calcul des déperditions thermiques commence ci-dessous et se poursuit pages 506 et 507 ; un exemple d'exécution suit, en pages 508 et 509.

PROGRAMME DE CALCUL DES DEPERDITIONS THERMIQUES

```
10 REM ** PROGRAMME DE CALCUL DES DEPERDITIONS THERMIQUES **
11 REM
12 REM
14 REM exécute en MICROSOFT BASIC
15 REM sur HHC (PANASONIC)
16 REM
18 REM
19 REM
20 REM
21 REM N.B. ici l'instruction standard LPRINT est remplacée
22 REM par PRINT #2 (2=numéro logique de l'imprimante)
23 REM
24 REM   liste des variables utilisées
25 REM   TI  = température intérieure
26 REM   TE  = température extérieure
27 REM   HA  = hauteur des pièces
28 REM   NP$ = nom de la pièce
29 REM   TM$ = type de mur (MB/MC)
30 REM   EM  = épaisseur du mur
31 REM   EX$ = exposition (N,S,E,O) du mur
32 REM   LM  = longueur du mur
33 REM   P   = coefficient d'exposition
34 REM   SI  = surface d'une fenêtre
35 REM   SF  = surface totale fenêtres du mur
36 REM   QF  = déperditions par fenêtres
37 REM   SM  = surface réelle du mur
38 REM   QM  = déperditions par le mur
39 REM   QT  = somme de QM+QF pour le mur
40 REM   QP  = somme des QT pour la pièce
41 REM   NU  = numéro de zone du sol
42 REM   L1  = côté 1 du sol
43 REM   L2  = côté 2 du sol
44 REM   SS  = surface du sol
45 REM   ST  = somme des SS
46 REM   QS  = déperditions par sol + plafond
49 REM   QA  = déperditions par aération
50 REM   T1  = total des déperditions = puissance
51 REM         de l'installation de chauffage
52 REM   T2  = somme des QA
53 REM   T3  = somme des QS
54 REM   T4  = somme des QT
55 REM
56 REM   initialisations
57 REM   tableaux en entrée
60 DIM E(3):REM épaisseur murs
70 DIM KB(3),KC(3):REM coefficients thermiques
80 E(1)=25
90 E(2)=38
100 E(3)=50
110 KB(1)=1.6
120 KB(2)=1.3
130 KB(3)=1
140 KC(1)=2.2
150 KC(2)=1.8
```

```
160 KC(3)=1.6
170 REM   autres initialisations
180 T1=0:T2=0:T3=0:T4=0
181 REM
182 REM   PROGRAMME PRINCIPAL
183 REM
190 INPUT "température intérieure :";TI
200 INPUT "température extérieure :";TE
210 INPUT "hauteur des pièces :";HA
211 REM
212 REM   boucle sur les pièces
220 PRINT "tapez le mot FIN pour sortir"
230 INPUT "nom de la pièce :";NP$
240 IF NP$="FIN"GOTO 390
250 QP=0
251 REM   boucle sur les murs d'une pièce
260 GOSUB 1000:REM routine MUR
270 IF C$=""GOTO 320
280 GOSUB 2000:REM routine FENETRE
290 GOSUB 3000:REM routine CHALEUR
300 QP=QP+QT
310 GOTO 260
311 REM
312 REM   fin du traitement de la pièce
320 GOSUB 4000:REM routine SOL
330 GOSUB 5000:REM routine AERATION
340 PT=QP+QS+QA
350 PRINT #2 "DEPERDITIONS THERMIQUES DANS LA PIECE ";NP$;" : ";PT
360 PRINT #2:PRINT #2
370 T1=T1+PT:REM cumul des pertes
380 GOTO 220
381 REM
382 REM   instruction d'impression
383 REM
390 PRINT #2"temperature extérieure : ";TE
400 PRINT #2"temperature intérieure : ";TI
410 PRINT #2"hauteur des pièces : ";HA
420 PRINT #2
430 PRINT #2"DEPERDITIONS TOTALES (cal/h) : "
440 PRINT #2
450 PRINT #2"murs et fenêtres : ";T4
460 PRINT #2"sols et plafonds : ";T3
470 PRINT #2"aération : ";T2
480 PRINT #2
490 PRINT #2"PUISSANCE DE L'INSTALLATION DE CHAUFFAGE : ";T1
500 END
501 REM
502 REM
503 REM   SOUS-PROGRAMME MUR
504 REM
1000 PRINT "sous-programme MUR":PRINT NP$
1010 PRINT "tapez seulement RETURN quand il n'y a plus de mur à traiter"
1020 INPUT "côté :";C$
1030 IF C$=""THEN GOTO 1300
1040 INPUT "type de mur (MB/MC) :";TM$
1050 IF TM$<>"MB"AND TM$<>"MC"THEN PRINT "erreur de type":GOTO 1040
1060 INPUT "épaisseur du mur (25/38/50) :";EM
1070 N=0
1080 FOR I=1 TO 3
1090 IF EM=E(I) THEN N=1:GOTO 1130
1100 NEXT I
1110 PRINT "erreur sur l'épaisseur"
1120 GOTO 1060
1130 IF TM$="MB"THEN K=KB(N):GOTO 1150
1140 K=KC(N)
1150 INPUT "exposition du mur (NORD,SUD,EST,OUEST) :";EX$
1160 INPUT "longueur du mur (metres) :";LM
1170 P=1
1180 IF EX$="NORD"THEN P=1.2
1190 IF EX$="OUEST"THEN P=1.1
1200 IF EX$="EST"THEN P=1.15
1201 REM
1210 PRINT #2"pièce : ";NP$
```

```
1220 PRINT #2"côté de la pièce : ";C$
1230 PRINT #2"K = ";K
1240 PRINT #2"coefficient d'exposition P = ";P
1250 PRINT #2"longueur du mur en mètres : ";LM
1300 RETURN
1310 REM
1311 REM   SOUS-PROGRAMME FENETRE
1312 REM
2000 PRINT "sous-programme FENETRE"
2010 PRINT NP$
2020 SF=0
2030 INPUT "surface de l'ouverture en m² (0 pour sortir) : ";SI
2040 IF SI=0 THEN GOTO 2070
2050 SF=SF+SI
2060 GOTO 2030
2070 QF=SF*5*P*(TI-TE)
2071 REM
2080 PRINT #2"surface totale fenêtres : ";SF
2090 PRINT #2"déperditions de chaleur par les fenêtres (cal/h): ";QF
2100 RETURN
2110 REM
2111 REM   SOUS-PROGRAMME CHALEUR
2112 REM
3000 SM=(LM*HA)-SF
3010 QM=SM*K*P*(TI-TE)
3020 QT=QM+QF
3021 REM
3030 PRINT #2"déperditions de chaleur par le mur : ";QM
3040 PRINT #2"deperditions totales (mur + fenêtres) : ";QT
3045 PRINT #2
3050 T4=T4+QT
3060 RETURN
3070 REM
3071 REM   SOUS-PROGRAMME SOL
3072 REM
4000 PRINT "sous-programme SOL"
4010 PRINT NP$
4020 ST=0
4030 QS=0
4040 PRINT "donnez un numéro de zone même si"
4050 PRINT "le sol n'a pas été subdivisé"
4060 INPUT "numéro de zone (0 pour sortir) :";NU
4070 IF NU=0 THEN GOTO 4220
4080 INPUT "longueur 1er côté : ";L1
4090 INPUT "longueur 2e côté : ";L2
4100 SS=L1*L2:REM surface de la zone
4110 ST=ST+SS:REM cumul surface de la pièce
4120 PRINT "choisissez le numéro voulu :"
4130 PRINT "0=pas de pertes (ni par le sol ni par le toit)"
4140 PRINT "1=déperditions par le sol"
4150 PRINT "2=déperditions par le toit"
4160 PRINT "3=pertes par le sol et par le toit"
4170 INPUT "votre choix :";C
4180 IF C=0 THEN GOTO 4060
4190 IF C>3 THEN PRINT "erreur":GOTO 4120
4200 QS=QS+SS*C*(TI-TE)
4210 GOTO 4060
4211 REM
4220 PRINT #2"autres déperditions"
4230 PRINT #2"pertes de chaleur,par le sol et/ou le toit : ";QS
4240 T3=T3+QS
4250 RETURN
4260 REM
4261 REM   SOUS-PROGRAMME AERATION
4262 REM
4263 REM   chaleur perdue pendant l'aération
4264 REM   = volume pièce * 0.3 * écart températures
5000 QA=ST*HA*0.3*(TI-TE)
5001 PRINT #2"pertes de chaleur en aération : ";QA
5015 PRINT #2
5020 T2=T2+QA
5030 RETURN
```

```
pièce : SALLE DE SEJOUR
côté de la pièce : BC
K = 1
coefficient d'exposition P = 1.15
longueur du mur en mètres : 3.5
surface totale fenêtres : 1.8
déperditions de chaleur par les fenêtres (cal/h) : 134.55
déperditions de chaleur par le mur : 130.065
déperditions totales (mur+fenêtres) : 264.615

pièce : SALLE DE SEJOUR
côté de la pièce : CD
K = 1
coefficient d'exposition P = 1
longueur du mur en mètres : 4
surface totale fenêtres : 2.52
déperditions de chaleur par les fenêtres (cal/h) : 163.8
déperditions de chaleur par le mur : 123.24
déperditions totales (mur+fenêtres) : 287.04

autres déperditions

pertes de chaleur par le sol et/ou le toit : 396.5
pertes de chaleur en aération : 178.425
DEPERDITIONS THERMIQUES DANS LA PIECE (SALLE DE SEJOUR) : 1126.58

pièce : SALLE DE BAINS
côté de la pièce : DE
K = 1
coefficient d'exposition P = 1
longueur du mur en mètres : 1.8
surface totale fenêtres : 1.2
déperditions de chaleur par les fenêtres (cal/h) : 78
déperditions de chaleur par le mur : 54.6
déperditions totales (mur+fenêtres) : 132.6

autres déperditions

pertes de chaleur par le sol et/ou le toit : 117
pertes de chaleur en aération : 52.65
DEPERDITIONS THERMIQUES DANS LA PIECE (SALLE DE BAINS) : 302.25

pièce : CUISINE
côté de la pièce : EF
K = 1
coefficient d'exposition P = 1
longueur du mur en mètres : 2.6
surface totale fenêtres : 2.52
déperditions de chaleur par les fenêtres (cal/h) : 183.8
déperditions de chaleur par le mur : 68.64
déperditions totales (mur+fenêtres) : 232.44

autres déperditions

pertes de chaleur par le sol et/ou le toit : 169
pertes de chaleur en aération : 76.05
DEPERDITIONS THERMIQUES DANS LA PIECE (CUISINE) : 477.49

pièce : CHAMBRE
côté de la pièce : FG
K = 1
coefficient d'exposition P = 1
longueur du mur en mètres : 3.5
surface totale fenêtres : 1.8
déperditions de chaleur par les fenêtres (cal/h) : 117
déperditions de chaleur par le mur : 113.1
déperditions totales (mur+fenêtres) : 230.1

autres déperditions
```

```
pertes de chaleur par le sol et/ou le toit : 338
pertes de chaleur au niveau : 152.1
DEPERDITIONS THERMIQUES DANS LA PIECE (CHAMBRE) : 720.2

température extérieure : 5
température intérieure : 18
hauteur des pièces     : 3

DEPERDITIONS TOTALES (cal/h) :

murs et fenêtres : 1146.785

sols et plafonds : 1020.5

aération         : 459.225

PUISSANCE DE L'INSTALLATION DE CHAUFFAGE : 2626.52
```

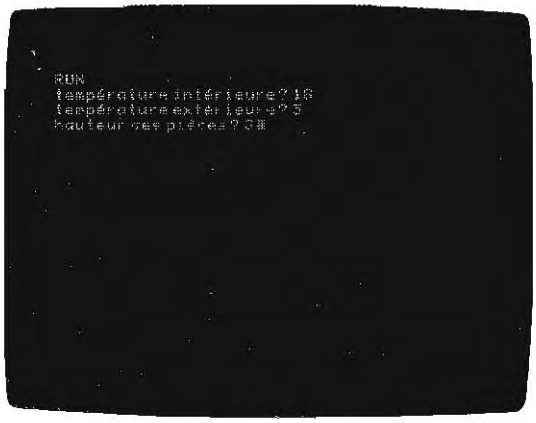
Lors de l'impression des résultats, le mot « calorie » désigne la « grande calorie » ou kilocalorie (1 kilocalorie = 1000 calories).
Les reproductions d'écran qui se trouvent pages 510 à 514 illustrent la partie interactive de l'exécution du programme : la phase de saisie des données. De même que ceux du listing, les exemples montrés dans cette séquence se rapportent tous à l'appartement dont le plan détaillé figure page 515.

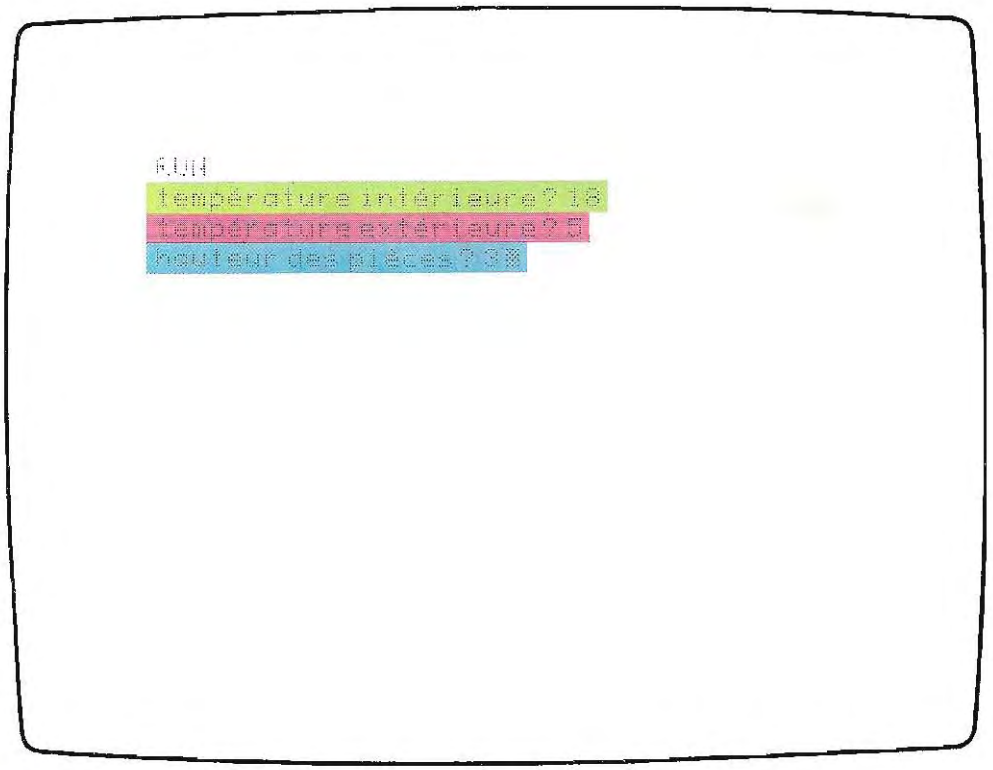
Ce cliché aux infrarouges révèle une isolation thermique insuffisante, voire inexistante.

AGA Infrared Systems/Science Photo Library

PROGRAMME DE CALCUL DES DEPERDITIONS THERMIQUES : SAISIE DES DONNEES (phase 1)

On lance l'exécution du programme par la commande RUN. L'interpréteur commence alors à traduire et à exécuter les lignes d'instructions, en partant de celle qui porte le plus petit numéro.





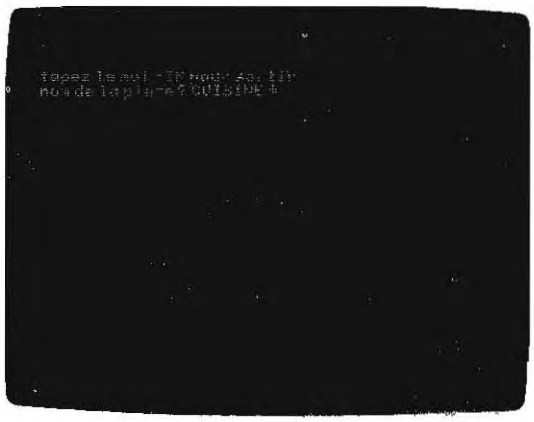
L'interpréteur exécute la ligne 190, qui demande une entrée au clavier (l'instruction INPUT est expliquée dans le chapitre consacré aux opérations d'E/S). On voit apparaître sur l'écran le message qui figure entre guillemets dans cette ligne du programme. Le point d'interrogation, ajouté par le système, indique que l'exécution est suspendue en attendant la donnée demandée.
L'opérateur tape 18 sur son clavier et valide ce nombre par la touche RETURN. La valeur entrée est alors affectée à la variable TI, comme le prévoit l'instruction 190.

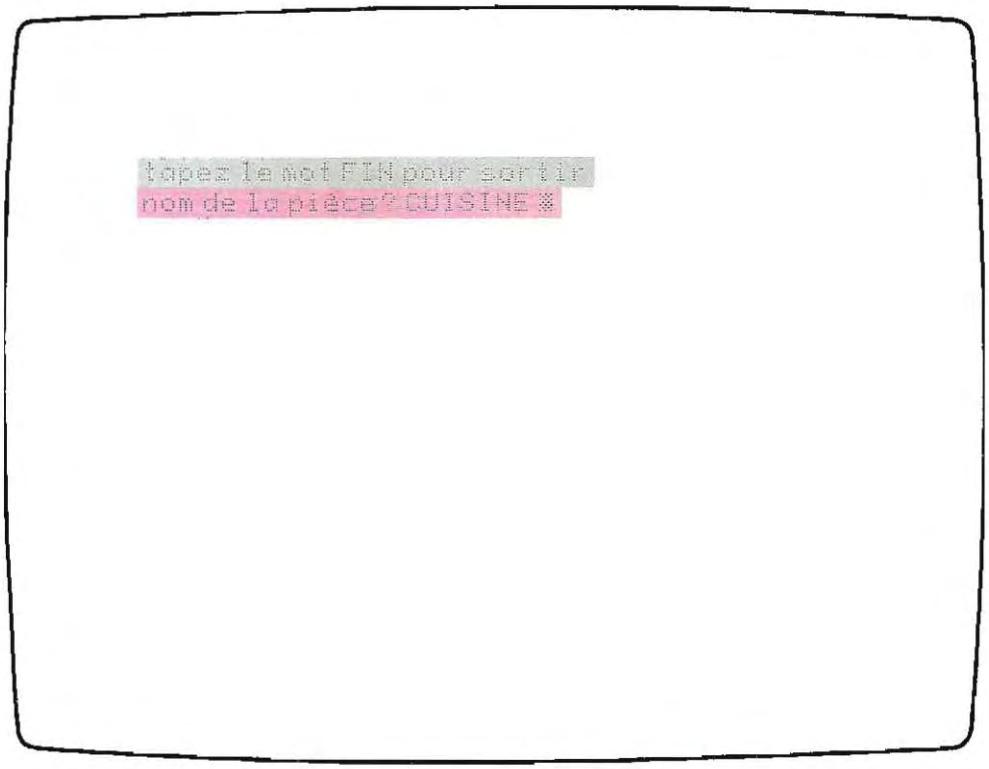
Le programme prévoit la saisie d'une nouvelle donnée (ligne 200). L'opérateur entre la valeur 5, qui est affectée à la variable TE.

Cette fois (ligne 210), c'est la hauteur des pièces qui est demandée (3 m). A l'instant où la photo a été prise, l'opérateur n'avait pas encore appuyé sur la touche RETURN, et le curseur clignotait donc encore à la fin de la dernière ligne.

PROGRAMME DE CALCUL DES DEPERDITIONS THERMIQUES : SAISIE DES DONNEES (phase 2)

Voici maintenant le moment de choisir la pièce dont on veut calculer les déperditions.





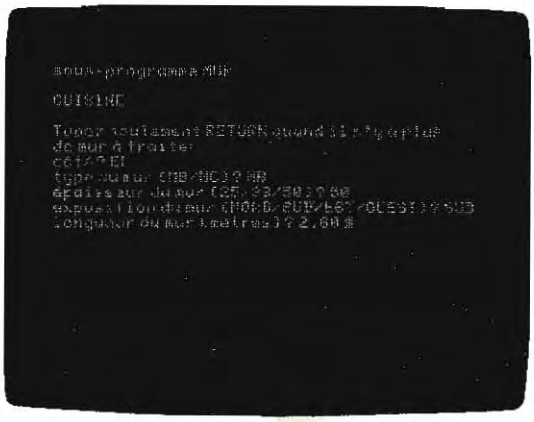
L'exécution de la ligne 220 entraîne l'affichage de ce message, qui indique à l'opérateur comment mettre fin à l'exécution (en tapant le mot FIN en guise de nom de pièce). Remarquons qu'il s'agit là de la seule façon logique et correcte d'interrompre la boucle du programme.

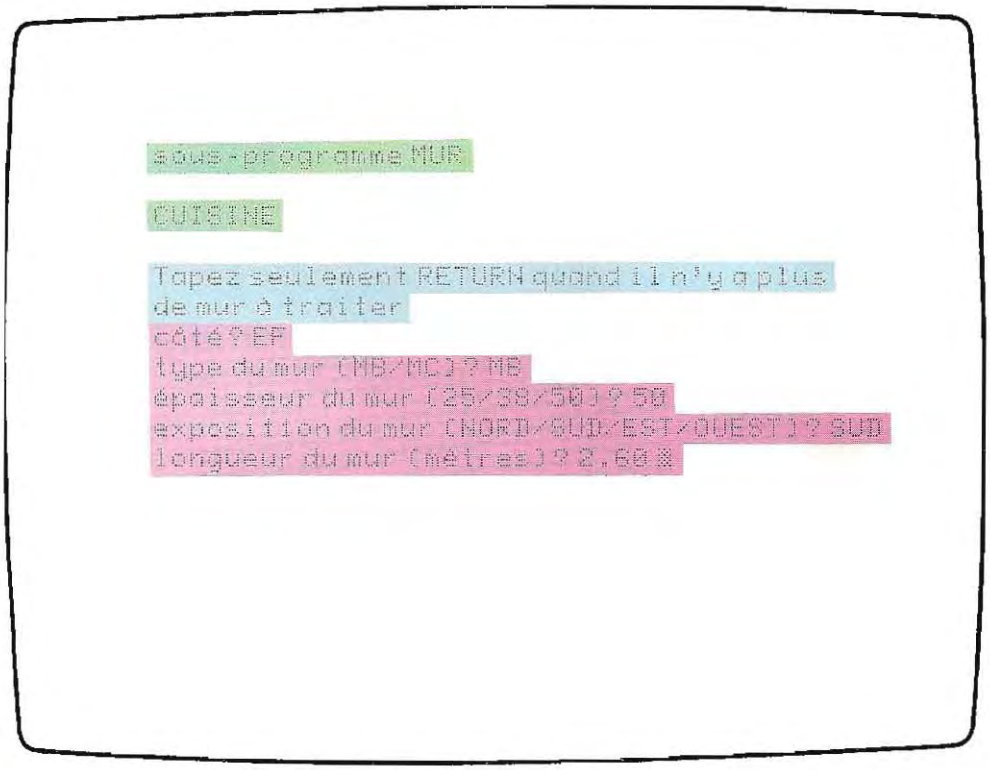
Cette méthode est très fréquemment employée dans les applications en mode interactif (dialogué). L'utilisateur doit suivre à la lettre les indications qui lui sont fournies, s'il veut que l'exécution se poursuive correctement. Il faut donc, bien sûr, que le programmeur pense à guider le mieux possible les opérateurs, par des messages comme celui-ci, et leur accorde le « droit à l'erreur » en prévoyant tous les contrôles nécessaires.
En effet, un programme interactif mal conçu sera très pénible à utiliser, et on risque, de plus, de perdre des données précieuses.

En ligne 230, le programme saisit le nom de la pièce dont les déperditions vont être calculées. L'opérateur tape le mot CUISINE, qui est affecté à la chaîne de caractères NP$.

PROGRAMME DE CALCUL DES DEPERDITIONS THERMIQUES : SAISIE DES DONNEES (phase 3)

Après la saisie des données communes à toutes les pièces (phase 1) et du nom de la première pièce à traiter (phase 2), l'instruction 260 appelle la routine 1000 (SOUS-PROGRAMME MUR).





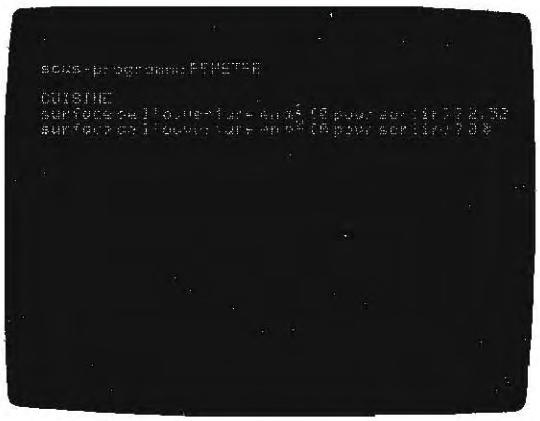
L'instruction 1000 provoque l'apparition de ce message à l'écran, ainsi que l'affichage du mot CUISINE, qui rappelle à l'opérateur quelle est la pièce en cours de traitement.

Ce dernier message indique à l'opérateur comment quitter la boucle de traitement des murs de la pièce. En effet, si la chaîne de caractères saisie est vide (l'opérateur a tapé seulement RETURN), on sort du sous-programme MUR, et on passe à la routine SOL.

Ici, l'opérateur entre successivement les informations qui lui sont demandées aux lignes 1020, 1040, 1060, 1150 et 1160. Remarquez que l'ordre de saisie des données est imposé par les messages qui accompagnent les instructions INPUT.

PROGRAMME DE CALCUL DES DEPERDITIONS THERMIQUES : SAISIE DES DONNEES (phase 4)

Lors de l'exécution du sous-programme 1000, la saisie d'une chaîne vide comme nom de mur, rend la main au programme principal, qui appelle alors le sous-programme SOL. Dans le cas contraire, on appelle ensuite la routine FENÊTRE, au cas ou le mur qui vient d'être traité contiendrait une ouverture.



Exécution de l'instruction 2010

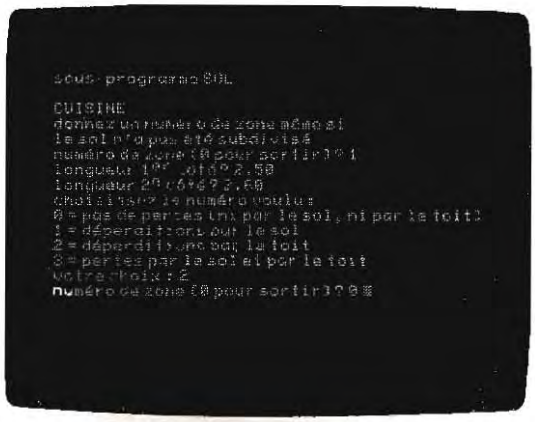
Le nom de la pièce est rappelé à l'opérateur.

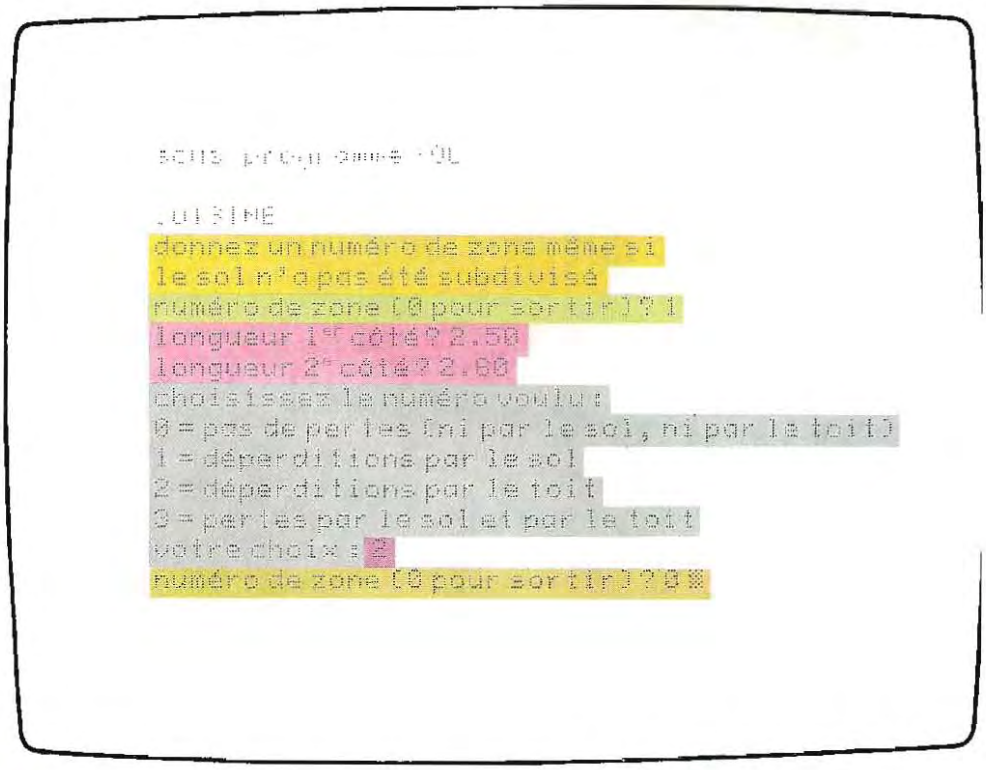
Ce message s'affiche à chaque passage par la boucle (saisies successives des surfaces des différentes fenêtres du mur). Il n'y a ici qu'une fenêtre (voir le plan de l'appartement).

La saisie du nombre 0 permet de quitter la boucle : on passe au calcul des déperditions de chaleur par les fenêtres (ligne 2070).

PROGRAMME DE CALCUL DES DEPERDITIONS THERMIQUES : SAISIE DES DONNEES (phase 5)

Le sous-programme SOL, appelé à la ligne 320 du programme principal, constitue l'étape suivante. Différents messages s'affichent à l'écran.





Les lignes 4040 et 4050, en affichant ce message, guident l'opérateur afin que l'exécution du programme se déroule correctement.
Quand une pièce a une forme complexe, il est nécessaire de découper sa surface en plusieurs rectangles ou carrés ; les dimensions de chacune de ces parties permettront de calculer séparément leurs surfaces respectives, dont la somme donnera la surface totale

L'opérateur devra attribuer un numéro de zone au sol de la pièce, même lorsqu'elle n'est pas subdivisée.

Saisie du numéro de zone.

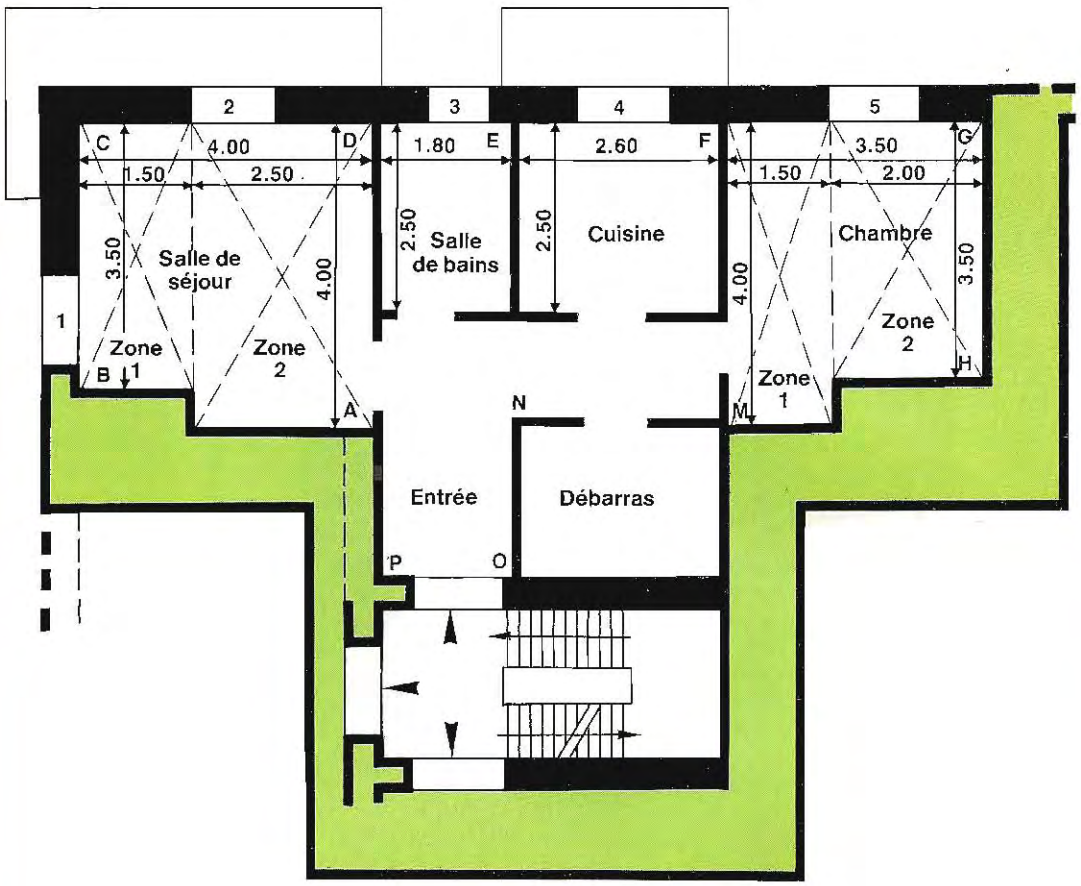
Saisie des longueurs des côtés d'une zone.

On demande maintenant à l'opérateur de préciser la situation de la zone considérée.

La réponse tapée au clavier signifie que, pour cette zone, la chaleur ne se dissipe que par le toit.

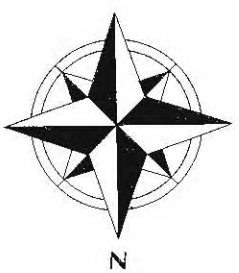
Les déperditions de chaleur sont calculées pour cette zone. L'instruction 4210 renvoie à la ligne 4060, pour traiter éventuellement une nouvelle ligne. La réponse de l'opérateur (0) détermine le retour au programme principal

PLAN DE L'APPARTEMENT



N.B. La chaleur ne se dissipe que par les murs BC, CD, DE, EF et FG, car les autres murs sont mitoyens avec d'autres appartements.

Légende: 1 = Fenêtre de 1.8 m²
2 = Porte-fenêtre de 2.52 m²
3 = Fenêtre de 1.2 m²
4 = Porte-fenêtre de 2 52 m²
5 = Fenêtre de 1.8 m²



Sol de la salle de séjour :	Zone 1 =	1.50 × 3.50
	Zone 2 =	2.50 × 4.00
Sol de la chambre	Zone 1 =	1.50 × 4.00
	Zone 2 =	2.00 × 3.50
Sol de la salle de bains	=	1.80 × 2.50
Sol de la cuisine	=	2.60 × 2.50

L'ordinateur à l'étable

On compte actuellement 210 millions de vaches dans le monde. La production laitière annuelle serait de 427 millions de tonnes, dont 5 millions de tonnes sous forme de lait en poudre, auquel il convient d'ajouter 7 millions de tonnes de beurre et 11 millions de tonnes de fromages. On imagine sans peine les problèmes que posent le transport et la distribution de telles quantités de denrées alimentaires à tous les gouvernements, et notamment à ceux de la Communauté Economique Européenne.

Confrontés aux mutations économiques et à l'augmentation incessante du coût global des aliments destinés au bétail, éleveurs et agriculteurs se voient contraints de planifier rigoureusement leur production. C'est pourquoi, à côté des vaches, apparaît aujourd'hui dans les fermes un nouveau pensionnaire : l'ordinateur.

Il représente, en effet, un outil précieux pour gérer les élevages de plus en plus importants des producteurs de lait. De nombreux autres éleveurs européens ont récemment abandonné ce secteur, encouragés par les subventions (prime à la reconversion) de la CEE, visant à résorber l'excédent de lait. Le cheptel de vaches laitières n'a pas diminué pour autant. Ainsi, en Grande-Bretagne, si les producteurs laitiers ont vu leur nombre baisser de 40 % depuis 1970, et ne sont plus aujourd'hui que 60.000 environ, l'exploitation moyenne compte 52 têtes de bétail contre 25 en 1965, et ce nombre ne cesse de croître.

La ***gestion*** *efficace d'un élevage passe nécessairement par la prévision de toutes les dépenses : frais généraux, investissements, nourriture et entretien. Dans ce but, un projet informatique a été mis au point, qui permet de prévoir un véritable calendrier du troupeau pour une année entière. Il est actuellement expérimenté au Royaume-Uni (où il a vu le jour), ainsi qu'aux Pays-Bas et dans les deux Irlandes. Les exploitants communiquent à un centre informatique situé en Grande-Bretagne, la production mensuelle de chaque vache, ainsi que des renseignements annexes (mises bas, accouplements...). L'ordinateur calculera, à partir de ces données, la production de chaque vache pendant le mois suivant, sa ration alimentaire journalière, ainsi que la date à laquelle cessera sa lactation.*

Des ***logiciels*** *comparables ont été conçus aux Etats-Unis, notamment avec le concours des University Agricultural Extension Services Centers. Dans l'un de ces centres, on détermine les caractéristiques de l'alimentation idéale pour l'hiver à venir. Les agriculteurs intéressés n'ont qu'à téléphoner pour entendre une voix synthétique (féminine !) leur fournir tous les renseignements voulus.*

Toujours aux Etats-Unis, l'Agricultural Economic Departement de l'Université du Michigan a mis en place une structure différente, fondée sur un système de temps partagé. On accède au Telplan par un téléphone un peu particulier, dont le clavier permet de communiquer des données à l'ordinateur en mode conversationnel.

Les producteurs adhérents pourront obtenir du Telplan une cinquantaine de ***plans d'action*** *adaptés à leur cas, qui les aideront, entre autres, à équilibrer les rations alimentaires de leur troupeau, à établir des budgets prévisionnels, ou qui les conseilleront dans leurs investissements. D'autres programmes du Telplan permettent de connaître les caractéristiques des différentes races bovines, et d'évaluer le bilan de production d'une vache.*

*Le gouvernement fédéral du Canada a ressenti depuis longtemps la nécessité de programmes de gestion mieux adaptés aux besoins des agriculteurs ; en 1965, a été créé le Canfarm, office à caractère régional et fédéral, visant le développement de l'agriculture sur l'ensemble du pays, à travers ses antennes régionales, ses établissements de formation et ses agences ouvertes au public. Les quelques 10.000 exploitants agricoles qui ont adhéré au Canfarm disposent aujourd'hui d'une large gamme de logiciels s'appliquant aux sujets les plus variés : les rations de fourrage, l'évolution prévisible des cours de la bourse, ou les derniers projets d'****élevages-modèle.***

Au Royaume-Uni également, plus de 20 % des producteurs laitiers utilisent l'informatique, d'une façon ou d'une autre, pour la gestion de leur exploitation. Ils envoient généralement leurs données par courrier au centre informatique de gestion auquel ils adhèrent. Ces données seront traitées sur de gros ordinateurs, ce qui explique les prix extrêmement élevés que pratiquent ces centres.

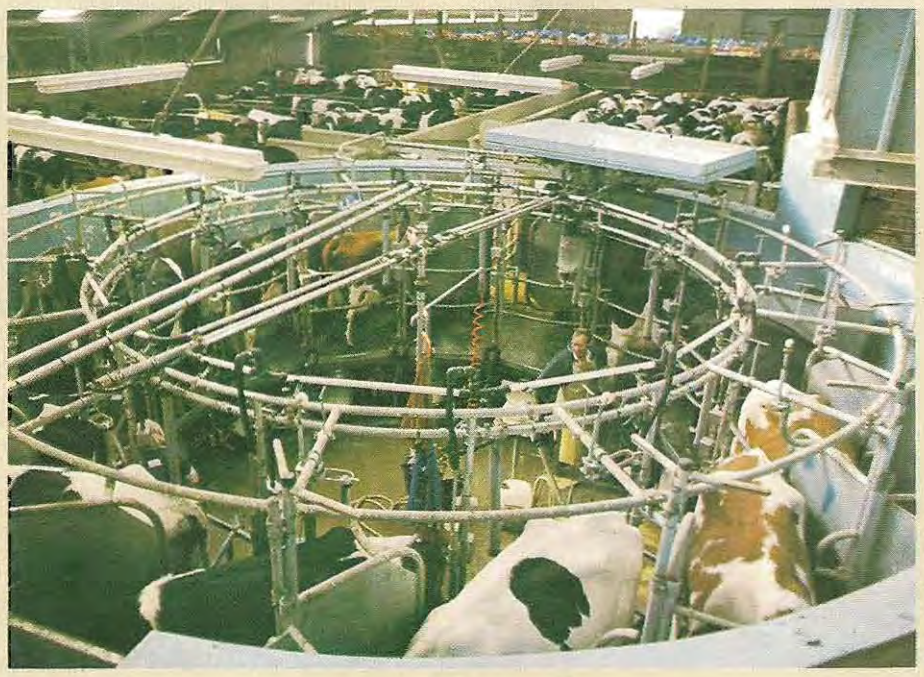
Mais une nouvelle tendance se dessine aujourd'hui. On commence à se tourner vers des systèmes de dimensions plus modestes, dont le prix raisonnable favorise la décentralisation. Le micro-ordinateur paraît réellement mieux adapté aux besoins des éleveurs laitiers, d'autant plus que la mise au point de «progiciels» spécifiques, ainsi que la connexion des ordinateurs à des dispositifs d'acquisition des données spécialement conçus, offrent des possibilités surprenantes. Parmi les systèmes informatiques de gestion à base de microprocesseurs, le Fullwood Ellesmere Electronics AFMS/80 est l'un des plus perfectionnés. C'est un logiciel d'application qui saisit et traite un grand nombre d'informations sur la gestion économique et technique d'un troupeau laitier.

Tous les animaux du troupeau portent un collier à médaille magnétique, qui permet à l'ordinateur de les identifier. Lorsque les vaches sont parquées en stabulation libre (et non attachées), on pourra ainsi repérer les déplacements de chacune d'entre elles, grâce au signal caractéristique de sa médaille: un ***récepteur*** *adéquat captera ce signal (généralement par une antenne), l'interprétera et le transmettra au microprocesseur chargé de l'identification. On peut ainsi savoir à tout moment si une vache a mangé, ou si elle a été traite, et tenir le compte des quantités de nourriture qu'elle ingère et de lait qu'elle produit.*

D'autre part, chaque box possède une fermeture ***magnétique****, qui ne s'ouvrira que devant le collier qui lui correspond. Le système, con-*

G.R. Roberts

En haut: installations d'un établissement de production de lait en poudre.



Alan Hutchison Library

Ci-contre: la salle de traite (en manège) dans une ferme modèle. Eleveurs et producteurs de lait doivent de plus en plus souvent informatiser leur exploitation pour survivre dans le monde moderne.

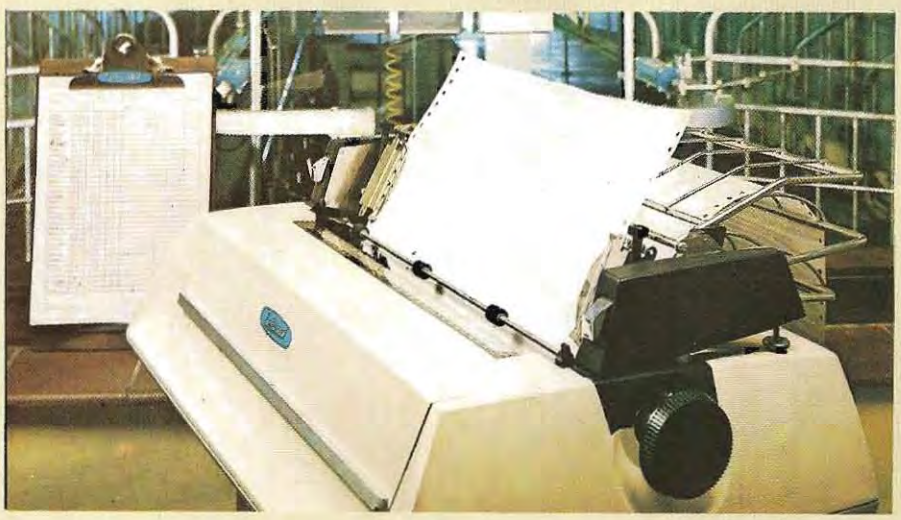
Les camions-citernes de cette laiterie moderne ont une capacité de 225.000 litres.

G.R. Roberts

Ci-dessous : le système Alfa Laval distribue aux vaches, quatre fois par jour et à heures fixes des quantités précises d'aliments concentrés. Une telle automatisation fait gagner du temps et supprime un travail pénible. De plus, sa régularité se traduit par une augmentation importante de la production de lait.

Alfa Laval

R.J. Fullwood & Bland Ltd.



R.J. Fullwood & Bland Ltd.

Ci-dessus : l'ordinateur fournit le calendrier des travaux à accomplir, la production de lait mensuelle et la fiche signalétique de chaque vache.

Ci-contre : l'imprimante édite des listings retraçant les informations reçues et les opérations exécutées.

naissant les besoins alimentaires de la vache, distribue donc automatiquement, cas par cas, la ration de concentré adéquate, en commandant une vis à aliment ou une boîte doseuse. Au cours de ses déplacements, la vache passe sur une balance automatique qui transmet son poids et son identité à l'ordinateur central.

Dans l'avenir, on envisage de remplacer le collier magnétique par un autre dispositif, qui serait implanté en permanence sur l'oreille ou la poitrine de l'animal, et établirait une liaison plus étroite encore avec l'ordinateur.

Le système Fullwood mesure la production effective de lait individuelle et globale au moyen d'un dynamomètre relié aux récipients dans lesquels le lait est recueilli. Le poids est lu par un microprocesseur qui le transmet à l'ordinateur. Ces mesures peuvent s'effectuer simultanément sur 32 postes de traite.

Mais l'assistance informatique à la conduite d'un troupeau ne se limite pas à l'alimentation et à la mesure de la production laitière : les besoins de chaque animal varient en fonction de son état de santé et de plusieurs autres facteurs.

Dans les exploitations équilibrées du Fullwood System, tous les contrôles et soins spécifiques dont doit bénéficier chaque vache sont mémorisés par le système.

Tous les boxes de l'étable sont équipés d'un témoin lumineux, qui s'allume si la vache, identifiée en entrant, est reconnue par le système comme devant subir un traitement particulier. Ainsi, si une vache souffre de mammite et doit recevoir des soins vétérinaires, l'ordinateur rappellera ce fait à l'éleveur, en lui envoyant les instructions appropriées. Plus tard, il demandera le prélèvement d'un échantillon de lait, qui sera analysé pour vérifier qu'il n'est pas contaminé par les médicaments administrés à l'animal.

Bien sûr, le rôle du « vacher » des temps modernes ne se borne pas à exécuter les instructions de l'ordinateur. Il s'agit plutôt d'un **dialogue** permanent, où le système, constamment enrichi par le savoir que lui transmet l'éleveur, jouera le rôle d'un assistant à la mémoire infaillible.

L'ordinateur propose à l'exploitant une liste journalière des travaux à accomplir – tels que les traitements à administrer aux animaux – ainsi qu'un calendrier des événements plus exceptionnels, comme les mises bas, les tarissements ou les dates d'insémination... D'autre part, un panorama hebdomadaire pourra indiquer les examens à faire subir aux vaches pleines et aux veaux nouveaux-nés, et rassemblera éventuellement des renseignements tels que le nombre de vaches qui sont venues s'ajouter au troupeau ou qui en sont sorties, le nombre d'animaux bénéficiant de soins particuliers et la répartition de la production de lait en fonction des différents usages prévus. Enfin, une fois par mois, ou plus souvent si nécessaire, on calculera le **rendement laitier** du troupeau (ou « moyenne d'étable »), en tenant compte de la production totale, de la taille du troupeau et du pourcentage de vaches pleines.

Plus les élevages sont importants, plus il devient difficile pour le vacher de connaître individuellement les bêtes et surtout de se rappeler leurs caractéristiques. Le système fournira à la demande une véritable fiche signalétique de chaque vache : son nom, son ascendance et sa descendance, sa production actuelle et celle des derniers mois, les soins qu'elle a subis, et bien d'autres données encore.

Il ne semble pas raisonnable d'envisager un tel investissement avec un troupeau de moins de 50 vaches. Avec un tel nombre, il faut compter environ 1.500 F par vache (uniquement pour l'alimentation automatique), et 300 à 400 F par vache pour la gestion technico-économique du troupeau. Quels que soient les futurs acheteurs de ces systèmes sophistiqués, la clé de leur réussite résidera dans la façon dont ils auront compris leur utilisation. L'ordinateur aidera précieusement l'éleveur et lui permettra de mieux travailler, de mieux gérer, de mieux comprendre, donc elle sera d'autant plus efficace que l'homme saura s'en servir avec intelligence.

L'ordinateur a déjà fait ses preuves dans de nombreux secteurs de l'économie, et son application à la production laitière constitue un nouvel exemple de réussite. Toutefois, là comme ailleurs, rien ne peut remplacer l'**expérience** acquise par les éleveurs et les agriculteurs au fil de leur pratique quotidienne. Sans le concours des agriculteurs, il aurait d'ailleurs été impossible de mettre au point des logiciels capables de répondre à leurs besoins spécifiques.

Les tableaux

Les variables d'un programme sont des noms symboliques associés par le système d'exploitation à des zones mémoire. Il est souvent pratique de désigner plusieurs zones par un même nom générique. Ainsi, pour définir un tableau (array), comme dans le programme de calcul des déperditions thermiques (page 505), on réserve plusieurs zones mémoire contiguës, représentées par le même nom. On accèdera ensuite à chacun des éléments en complétant ce nom par l'indice (numéro d'ordre) de l'élément dans le tableau. En début de programme, on spécifiera le nom que le système doit affecter à cette zone de mémoire et le nombre d'éléments qu'elle contiendra (taille à réserver). L'instruction de dimensionnement est la suivante :

DIM NOM (N)

DIM (dimension) représente le code opération qui demande au système de réserver une partie de la mémoire pour le tableau, le temps de l'exécution du programme. Il est suivi du nom de la variable (ici, NOM) et du nombre d'éléments (N) qui est de type entier et s'appelle la « dimension » du tableau. L'instruction DIM TABL(20), par exemple, réserve une partie de mémoire pouvant contenir 20 éléments, et lui associe le nom TABL.

Pour accéder à un élément, on le désignera par son indice (c'est-à-dire son rang dans le tableau). Ainsi, l'instruction

R=6.8*TABL(4)

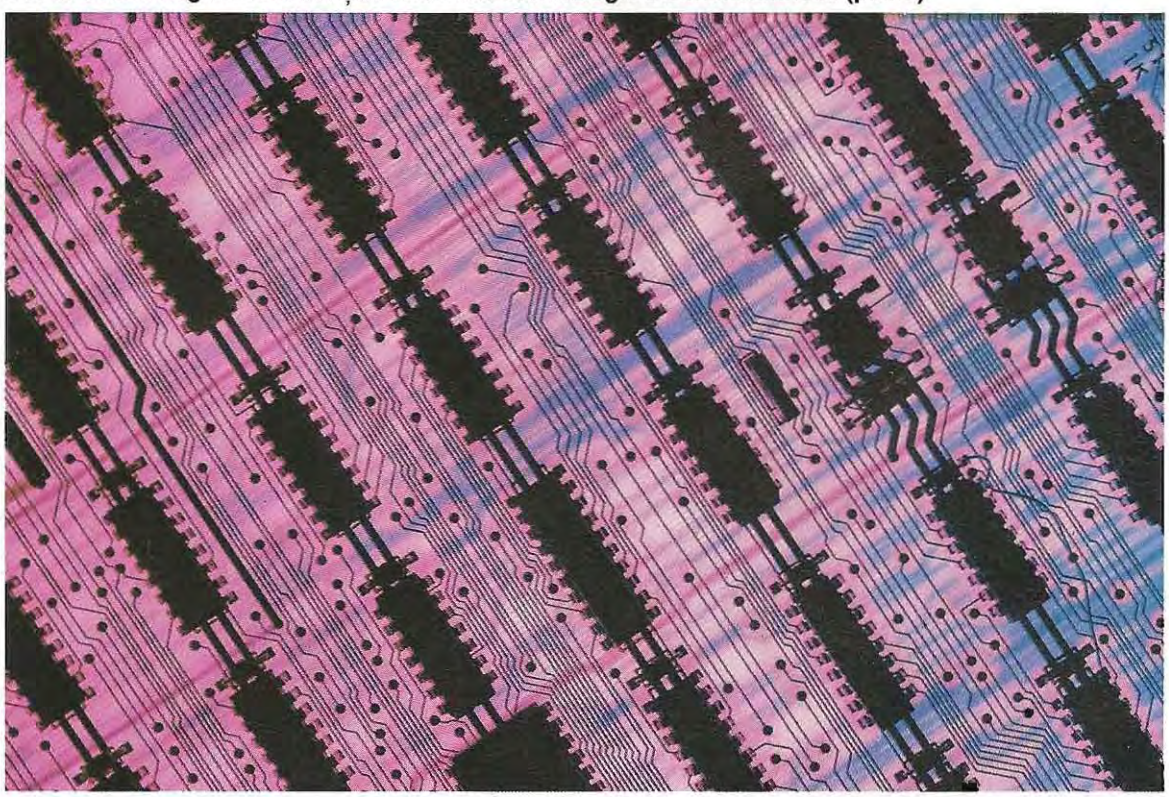
extrait la valeur située dans la quatrième case de la zone de variables TABL, la multiplie par 6.8 et affecte le résultat à R.

Il faut être très attentif à la numérotation des éléments d'un tableau : dans la plupart des Basic, le système « compte » en effet à partir de 0, et le premier indice n'est donc pas 1 (TABL(1)) représente alors le deuxième élément). Certains interpréteurs ou compilateurs Basic offrent au programmeur la possibilité de faire commencer la numérotation à 1, avec l'instruction OPTION BASE 1.

La coexistence de ces deux bases de numérotation des tableaux est expliquée dans le chapitre sur l'Assembleur.

Enfin, insistons sur le fait qu'un tableau est une variable au même titre qu'une autre : elle

Ces circuits intégrés sont conçus selon une technologie bidimensionnelle (plane).



Marka

possède donc un type déterminé.
Tous ses éléments doivent être homogènes : il est impossible, par exemple, de stocker dans un même tableau des entiers et des nombres en simple précision, ou bien des chaînes de caractères et des valeurs numériques.
Les différents types possibles, ainsi que les déclarations implicites (DEFINT, par exemple) seront les mêmes que pour les constantes ou les variables ordinaires, et toucheront alors tous les éléments du tableau.

Organisation des tableaux

Dans tous les programmes qui ont été présentés jusqu'ici, chaque variable possédait un nom propre et était mémorisée dans une zone réservée à elle seule, dont la longueur (nombre d'octets) dépendait du type de la variable elle-même : deux octets pour un entier, un octet pour chacun des caractères d'une chaîne, etc. Pour évaluer la place mémoire occupée par un tableau, il suffit de multiplier le nombre d'éléments qu'il peut contenir (dimension) par la longueur (en octets) d'un élément. Les instructions suivantes :

```
10 OPTION BASE 1
20 DEFINT A-K
30 DIM AR(3)
```

définissent un tableau nommé AR, qui contiendra trois éléments de type entier (l'instruction 20 fixe comme entières toutes les variables d'initiale comprise entre A et K; le tableau AR sera donc entier). La numérotation débute à 1, et le tableau occupera 6 octets en mémoire (2 par élément).
En Basic, il est possible de déclarer implicitement un tableau. Si l'on mentionne, au cours d'un programme, une variable indicée (tableau) sans l'avoir dimensionnée au préalable avec l'instruction DIM, le système lui affectera automatiquement une zone de mémoire. Dans l'exemple précédent, la ligne 30 qui déclare et dimensionne à 3 le tableau AR aurait pu être omise : à l'exécution, dès que le système aurait rencontré une instruction se référant à la variable indicée AR(N), il aurait réservé une zone de mémoire pour le tableau. Par exemple, l'instruction AR(3) = 4.7 aurait automatiquement créé en mémoire un tableau AR, avant d'effectuer l'affectation demandée (le troisième élément prend la valeur 4.7).
Ce mode de déclaration automatique possède cependant ses limites : le Basic Standard réservera systématiquement 10 emplacements pour un tableau dimensionné implicitement. L'instruction précédente [AR(3) = 4.7] sera correcte, mais AR(132) = 4.7 ne pourra pas dimensionner AR à 132 éléments. Dans ce cas, le programmeur devra écrire plus haut l'instruction : DIM AR(132).
Les tableaux de chaînes alphanumériques se déclarent et se dimensionnent de la même manière. La longueur d'une variable définie comme chaîne peut varier entre 0 et 128 caractères (ou 256 sur certains systèmes). Le système attribue automatiquement à la variable la place maximale (128 octets).
L'instruction DIM A$(3) réservera une zone de mémoire pour contenir trois chaînes distinctes, soit un encombrement physique de 128 × 3 = 384 octets au total.
Ainsi, le programme suivant :

```
10 OPTION BASE 1
20 DIM A$(3)
30 A$(1)="Première chaîne"
40 A$(2)="Deuxième"
50 A$(3)="Dernière"
60 END
```

crée en mémoire un tableau de trois chaînes [DIM A$(3)] de 128 octets chacune. La première contient 15 caractères (ligne 30 ; il faut tenir compte de l'espace entre les deux mots), la seconde 8 caractères (ligne 40) et la troisième également. L'occupation effective est donc de 15 + 8 + 8 = 31 octets, alors que 384 sont réservés. Toute variation de longueur de l'une des chaînes modifiera la longueur totale : si l'on pose maintenant A$(3)="Dernière chaîne", 38 octets seront occupés (7 caractères ont été ajoutés).
Il ressort clairement de cet exemple que la méthode utilisée (réservation systématique de 128 octets pour chaque chaîne) occasionne parfois un « gaspillage » de la place mémoire. Sur certains systèmes, il est possible d'imposer aux chaînes une longueur maximale, c'est-à-dire de les dimensionner.
Voici une caractéristique du Basic, qui donne une signification particulière à l'instruction

DIM, lorsqu'elle est appliquée à une variable de type chaîne. En effet, DIM CH$ [8] réservera 8 octets pour la chaîne CH$. La même formulation s'appliquera aux tableaux de chaînes : ainsi, l'instruction DIM A$(3) [8] réserve 24 octets pour trois chaînes, chacune de huit caractères au maximum. Dans ce cas, l'instruction 30 du programme ci-dessus provoquerait une erreur, en cherchant à affecter à A$(1) une chaîne de plus de huit caractères. Précisons toutefois que, comme la notion de « sous-chaînes », le dimensionnement des variables alphanumériques n'est possible que sur les systèmes les plus perfectionnés. Nous reviendrons sur ce point dans le chapitre consacré aux variantes du Basic (par rapport au Basic standard) proposées sur les différents micro-ordinateurs.

Tableaux à une et à N dimensions

L'instruction DIM AR(3) crée un tableau de trois éléments successifs : il n'a qu'une dimension.
Cette notion de dimension devient plus claire si l'on assimile les tableaux à des figures géométriques : une ligne n'a qu'une dimension, une figure plane en a deux (longueur et largeur) et un solide trois (longueur, largeur et hauteur). De même, un tableau sera organisé selon une dimension, que nous pourrons appeler sa « longueur », ou bien deux (sa longueur et sa largeur). S'il possède trois dimensions, on parlera de sa profondeur.
Le langage informatique n'emploie pas ces termes géométriques pour désigner les dimensions d'un tableau. Lorsque ce dernier possède deux dimensions, les informaticiens le découpent en lignes et colonnes, chaque élément se trouvant à l'intersection d'une ligne et d'une colonne. Quant à la troisième dimension (qui correspond à la profondeur), on l'appellera un plan (ou une page). Les deux premières dimensions définissent des pages divisées en lignes et en colonnes, qui se succèderont le long de la troisième dimension.
Pour stocker simplement une série de données, un tableau à une dimension est amplement suffisant. Ainsi, pour mémoriser les dépenses familiales quotidiennes pendant un mois, on dimensionnera un tableau à 31 éléments, le premier contenant le total des dépenses du premier jour du mois, le second celui du deuxième et ainsi de suite. Par contre, un tableau à deux dimensions sera nécessaire pour réaliser une table de multiplication : on stockera le produit I*J à l'intersection de la ligne I et de la colonne J. Ensuite, pour connaître le produit des nombres A et B, on lira le contenu de la case (A,B) du tableau. Les tableaux à trois dimensions seront indispensables pour certaines applications de gestion, mais leur emploi se justifie rarement.
La page 523 représente schématiquement trois tableaux ayant respectivement 1, 2 et 3 dimensions. Voici les déclarations correspondantes :

- DIM A(4)
  Tableau à une dimension, contenant quatre éléments
- DIM B(3,5)
  Tableau à deux dimensions, contenant $3 \times 5 = 15$ éléments, répartis sur 3 lignes et 5 colonnes
- DIM C (3,3,5)
  Tableau à trois dimensions, contenant $3 \times 3 \times 5 = 45$ éléments, répartis sur 3 pages organisées en 3 lignes et 5 colonnes chacune.

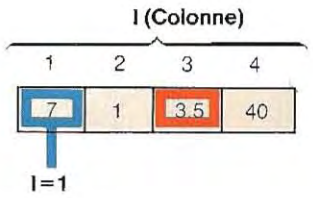
Un tableau peut posséder plus de trois dimensions (il n'existe plus de support géométrique intuitif dans ce cas !) mais il devient alors très lourd à manier. Il est généralement préférable d'organiser ses données selon plusieurs tableaux de trois noms différents. De plus, les opérations que devra effectuer le système d'exploitation pour manipuler les tableaux seront toujours plus simples et plus vite exécutées lorsque le nombre de dimensions est plus réduit.
En outre, beaucoup d'interpréteurs n'acceptent qu'un nombre limité de dimensions.

Instructions d'affectation sur des tableaux

Pour affecter des valeurs aux différents éléments d'un tableau, on procèdera de la même façon que pour les variables ordinaires : soit avec le symbole d'affectation directe =, soit par l'instruction DATA.
Voici, par exemple, un programme qui affecte aux éléments du tableau A les valeurs 7, 1, 3.5 et 40 (voir page 523, schéma du haut) :

QUELQUES EXEMPLES DE VARIABLES STRUCTUREES (TABLEAUX)

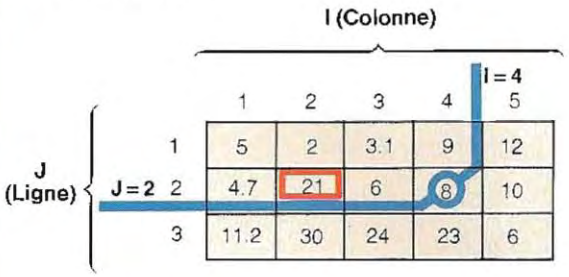
Une dimension



DIM A(4)
Exemple d'affectation :
A(3)=3.5

Exemple de lecture :
Si l'on pose I=1 alors A(I)=7

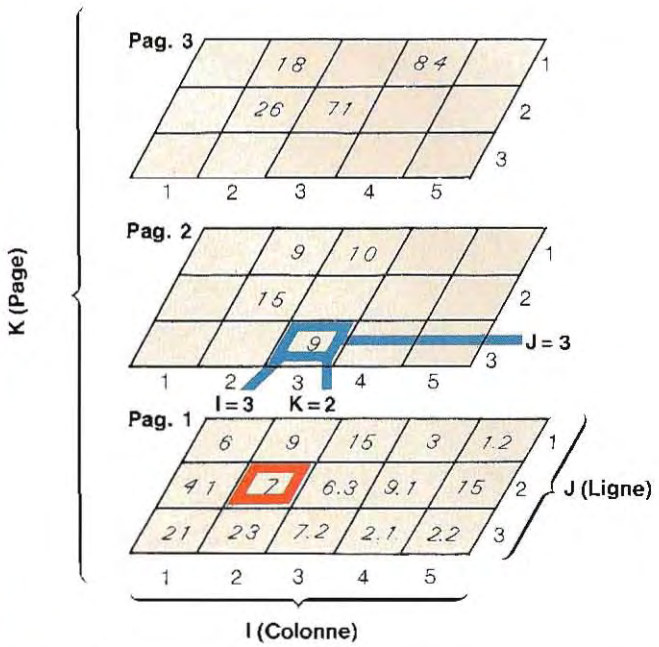
Deux dimensions



DIM B(3,5)
Exemple d'affectation :
A(2,2)=21

Exemple de lecture :
Si l'on pose J=2 et I=4 alors A(J,I)=8

Trois dimensions



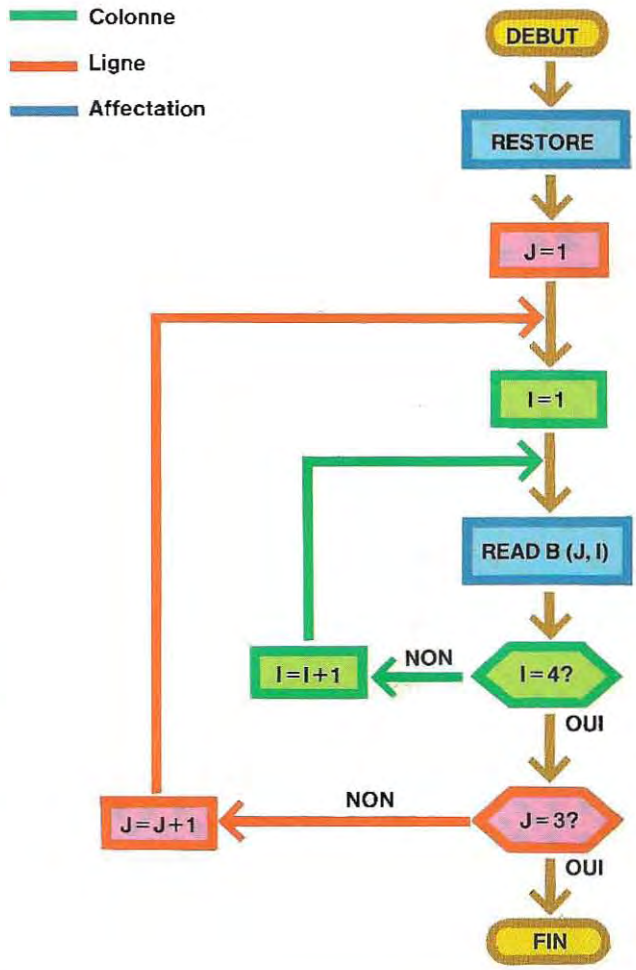
DIM C(3,3,5)

On peut considérer le premier indice comme le numéro de page, et les deux autres comme les numéros de ligne et de colonne dans la page.

Exemple d'affectation :
A(1,2,2,)=7

Exemple de lecture :
Si l'on pose I=3,J=3 et K=2 alors A(K,J,I)=9

REMPLISSAGE D'UN TABLEAU B(3,4) A DEUX DIMENSIONS (ORGANIGRAMME)




```
10 OPTION BASE 1
20 ' La numérotation commence à 1
30 DIM A(4)
40 A(1)=7:A(2)=1:A(3)=3.5:A(4)=40
50 END
```

Le même résultat sera obtenu avec les instructions suivantes :

```
10 OPTION BASE 1
20 ' La numérotation commence à 1
30 RESTORE 70
40 FOR I=1 TO 4
50 READ A(I)
60 NEXT I
70 DATA 7,1,3.5,40
80 END
```

La boucle constituée par les lignes 40, 50 et 60 affecte, l'une après l'autre, les valeurs contenues dans le DATA (ligne 70) à la case d'indice I du tableau A. La première donnée du DATA (7), est lue quand I = 1, et A(1) prend donc la valeur 7. De même, quand I vaut 2, la valeur 1 est affectée au deuxième élément du tableau [A(2) = 1] et ainsi de suite.

Vous trouverez pages 524 et 525 des organigrammes illustrant le déroulement logique du remplissage de tableaux à plusieurs dimensions.

Les listings correspondants se trouvent pages 526 et 527.

De la même manière, on peut également remplir un tableau de chaînes alphanumériques

au moyen de l'instruction DATA :

```
10 OPTION BASE 1
20 DEFINT I-N
30 DIM A$(3)
40 RESTORE
50 FOR I = 1 TO 3
60 READ A$(I)
70 NEXT I
80 DATA "Première ligne"
```

REMPLISSAGE D'UN TABLEAU A(3,3,5) A TROIS DIMENSIONS (ORGANIGRAMME)

REMPLISSAGE D'UN TABLEAU BIDIMENSIONNEL (LISTING)

```
10 ' ** PROGRAMME : TABLEAU BIDIMENSIONNEL **
15 '
20 '              FICHIER = BIDIM
25 '
30 OPTION BASE 1
40 DIM A(3,4) ' Déclaration du tableau
50 RESTORE 110
60 FOR J = 1 TO 3 ' Lecture
70 FOR I = 1 TO 4
80 READ A(J,I)
90 NEXT I
100 NEXT J
110 DATA 11,12,13,14,21,22,23,24,31,32,33,34
120 '
130 FOR J = 1 TO 3 ' Ecriture
140 LPRINT "LIGNE: ";J
150 FOR I = 1 TO 4
160 LPRINT A(J,I);
170 NEXT I
180 LPRINT ' Saut de ligne
190 NEXT J
200 END
```

```
LIGNE: 1
 11  12  13  14
LIGNE: 2
 21  22  23  24
LIGNE: 3
 31  32  33  34
```

REMPLISSAGE D'UN TABLEAU TRIDIMENSIONNEL (LISTING)

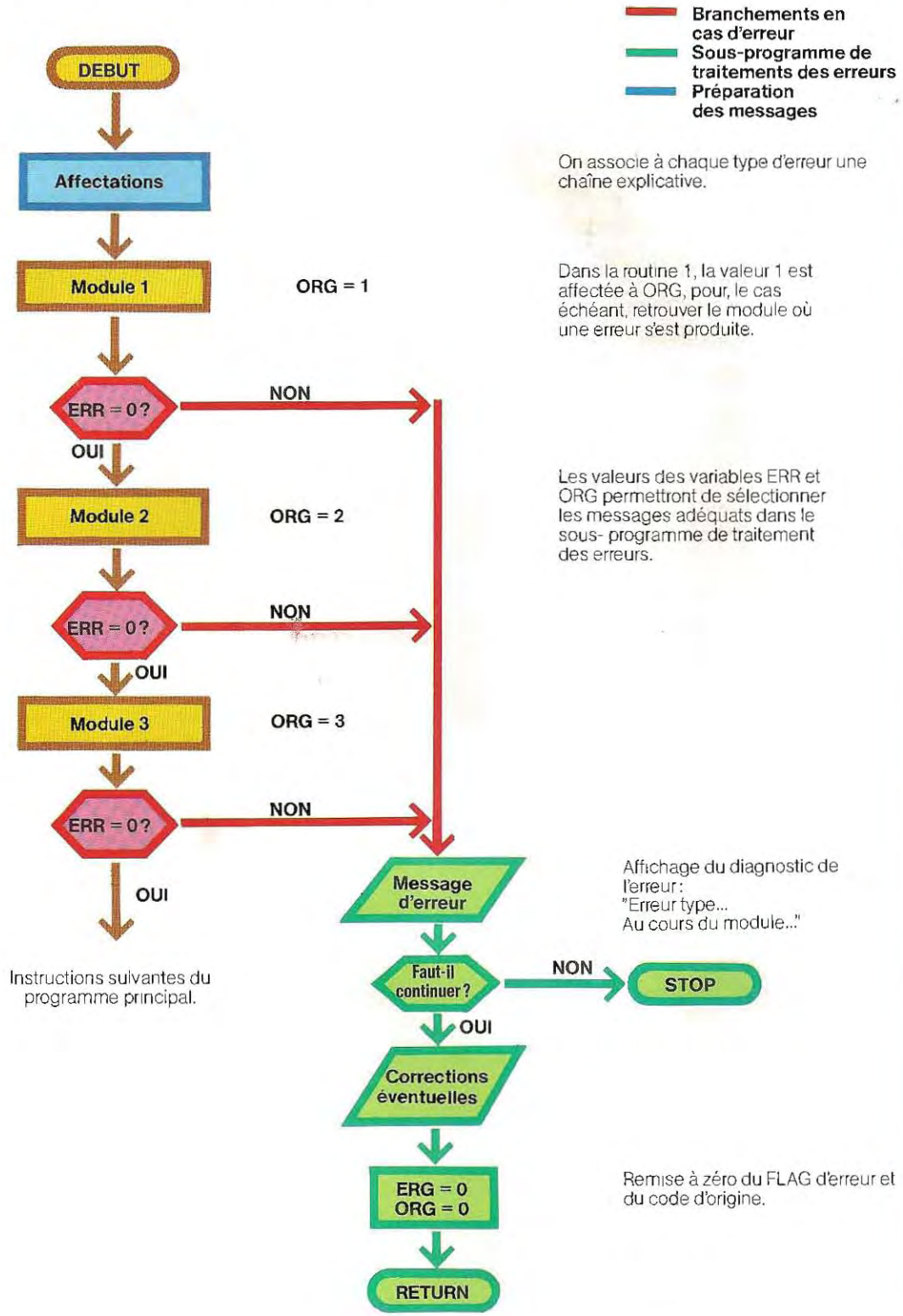
```
10 ' ** PROGRAMME : TABLEAU TRIDIMENSIONNEL **
15 '
20 '              FICHIER = TRIDIM
25 '
30 OPTION BASE 1
40 DIM A(3,3,4) ' Déclaration du tableau
50 RESTORE 140
60 '
70 FOR K = 1 TO 3
80 FOR J = 1 TO 3
90 FOR I = 1 TO 4
100 READ A(K,J,I)
110 NEXT I
120 NEXT J
130 NEXT K
140 DATA 11,12,13,14,21,22,23,24,31,32,33,34
150 DATA 51,52,53,54,61,62,63,64,71,72,73,74
160 DATA 41,42,43,44,81,82,83,84,91,92,93,94
165 '
170 FOR K = 1 TO 3          ' Ecriture
180 LPRINT " PAGE :  ";K
190 FOR J = 1 TO 3
200 LPRINT "LIGNE:  ";J
210 FOR I = 1 TO 4
220 LPRINT A(K,J,I);
230 NEXT I
240 LPRINT
250 NEXT J
260 LPRINT
270 NEXT K
280 END
```

```
PAGE: 1
LIGNE: 1
11  12  13  14
LIGNE: 2
21  22  23  24
LIGNE: 3
31  32  33  34

PAGE: 2
LIGNE: 1
51  52  53  54
LIGNE: 2
61  62  63  64
LIGNE: 3
71  72  73  74

PAGE: 3
LIGNE: 1
41  42  43  44
LIGNE: 2
81  82  83  84
LIGNE: 3
91  92  93  94
```

```
90 DATA "Deuxième"
100 DATA "Dernière"
110 END
```

Ce programme affecte aux éléments du tableau A$ les trois chaînes suivantes :
A$(1) = "Première ligne"
A$(2) = "Deuxième ligne"
A$(3) = "Dernière".
Un tableau de chaînes de caractères sera très utile pour la gestion des messages d'erreur. Au cours de l'exécution d'un programme, différentes erreurs peuvent être diagnostiquées ; certaines sont « rattrapables » par l'opérateur, surtout en phase de saisie de données ou de paramètres qui doivent ensuite servir au traitement.
Nous avons vu précédemment qu'en mode interactif, il est indispensable de signaler les erreurs éventuelles à l'opérateur, en émettant des messages qui apparaîtront à l'écran et lui expliqueront clairement la conduite à suivre pour les corriger.
On peut insérer ces messages d'erreur à n'importe quel endroit du programme, au fur et à mesure des besoins ; mais ceux-ci devront alors être répétés chaque fois qu'une même erreur peut se produire, et le programme principal va s'allonger et devenir illisible.
Il est beaucoup plus pratique de regrouper tous ces messages dans un tableau. Un sous-programme de diagnostic et d'affichage sera appelé dès qu'une erreur quelconque aura été commise.
Le code numérique affecté à l'erreur permet alors de sélectionner le message approprié dans le tableau.
En outre, il peut être intéressant de faire figurer à côté du message (destiné à l'utilisateur), un symbole qui indique le type d'erreur en cause.
Le travail de maintenance du programmeur se trouvera, dès lors, grandement facilité, grâce à une structuration optimisée du logiciel (ensemble des programmes).
En page 528, se trouve l'organigramme simplifié d'un programme construit de cette façon.
Les opérations à exécuter sont réparties en trois modules (routines), qui contiennent chacun des instructions de contrôle et de validation des données à traiter.
Si celles-ci ne répondent pas aux critères requis, une erreur sera diagnostiquée, et le code numérique correspondant sera affecté à la variable ERR, qui, dans le cas contraire, reste initialisée à la valeur 0. D'autre part, le numéro du module où l'erreur a été commise est affecté à la variable ORG.
Ainsi, lorsqu'à un moment donné on entre dans le module de traitement des erreurs avec les valeurs : ERR = 4, ORG = 2, on sait qu'il est survenu une erreur de type 4, au cours de l'exécution du module 2 (c'est le programmeur qui définit lui-même les erreurs et leurs numéros). On pourra alors afficher le message voulu. Les diagnostics d'erreur et les noms des modules sont déterminés dans le bloc « Affectations ».

STRUCTURE D'UN PROGRAMME POUR PREVOIR ET CONTROLER LES ERREURS



Pages 529 et 530, est listé un programme simulant la logique que nous venons d'exposer; les routines 1000, 2000 et 3000 demandent à l'utilisateur l'erreur qu'il veut générer et en produisent les résultats correspondants (pages 530 et 531).

Application des tableaux. Les variables structurées servent très souvent en programmation. L'organisation des données sous forme de tableaux simplifie considérablement l'écriture du programme, et le rend plus lisible.

Dans les pages qui suivent, nous présenterons deux applications typiques.
La première concerne l'utilisation de tableaux pour effectuer des calculs sur des variables numériques possédant un nombre quelconque de chiffres significatifs. La seconde nous donnera l'occasion d'introduire certaines notions de statistique appliquée, qui devraient faire partie du bagage technique de tout programmeur. La connaissance de ces notions n'est toutefois pas indispensable à la poursuite de notre étude.

PROGRAMME DE SIMULATION DES ERREURS

```
10 ' **PROGRAMME DE SIMULATION DES ERREURS**
20 '
30 '            PROGRAMME PRINCIPAL
40 '            FICHIER = SIMERR
50 '
60 ' **DECLARATIONS ET AFFECTATIONS **
70 OPTION BASE 1
80 DEFINT A-R          ' Les variables d'initiales comprises
90 '                     entre A et R sont entieres
100 DIM D$(4)          ' D$ contient quatre chaines
102 '                    de description des diagnostics
110 DIM R$(3)          ' Les trois chaines R$ contiennent, en entier,
112 '                    les noms des sous-programmes
120 RESTORE 190        ' Pointer au premier DATA (diagnostics)
130 FOR I = 1 TO 4     ' Lecture des chaines D$
140 READ D$(I)
150 NEXT I
160 '
170 ' **Les DATA suivants contiennent des diagnostics quelconques
172 '    a titre d'exemple.
190 DATA "Erreur de lecture ou d'ecriture sur disque - "
200 DATA "Choix de donnees non valides - "
210 DATA "Choix d'une option non prevue - "
220 DATA "Le mot de passe n'est pas reconnu - "
230 '
240 RESTORE 320        ' Pointer au deuxième DATA (noms des sous-programmes)
250 FOR I = 1 TO 3     ' Lecture des chaines R$
260 READ R$(I)
270 NEXT I
280 '
290 ' **Les noms sont donnes a titre d'exemple. En realite,
300 '    ces sous-programmes n'executent aucune fonction
310 '
320 DATA "Premier sous-programme"
330 DATA "Calcul interets"
340 DATA "Sous-programme d'essai"
350 '
360 ' **Les instructions RESTORE aux lignes 120 et 240
370 '    ne sont pas absolument necessaires
380 ERR = 0 : ORG = 0          ' Initialisation des flags
390 GOSUB 1000                 ' Appeler le premier sous-programme
400 IF ERR <> 0 GOSUB 9000     ' En cas d'erreur, appeler la ligne 9000
410 GOSUB 2000                 ' Deuxième sous-programme
420 IF ERR <> 0 GOSUB 9000
430 GOSUB 3000                 ' Troisieme sous-programme
440 IF ERR <> 0 GOSUB 9000
480 ' **FIN DU PROGRAMME
490 '
500 END
```

```
1000 ' ** ENTREE DANS LE PREMIER SOUS-PROGRAMME
1010 ' Demande l'erreur a simuler et retourne au programme principal
1020 ORG = 1                ' FLAG D'ORIGINE DE L'ERREUR
1030 INPUT "Entrer le code de l'erreur a simuler ";ERR
1040 RETURN
1050 ' Les sous-programmes 2000 et 3000 sont semblables : seul ORG change
1060 '
2000 ' ** DEUXIEME SOUS-PROGRAMME
2010 ORG = 2
2020 INPUT "Entrer le code de l'erreur a simuler ";ERR

2030 RETURN
3000 ' ** TROISIEME SOUS-PROGRAMME
3010 ORG = 3
3020 INPUT "Entrer le code de l'erreur a simuler ";ERR
3030 RETURN
9000 ' ** SOUS-PROGRAMME DE TRAITEMENT DES ERREURS
9010 LPRINT "Erreur n. : ";ERR
9020 LPRINT "Provenance : ";ORG
9030 '
9040 ' ** Les erreurs prevues sont comprises entre 1 et 4 (voir instructions 190-220)
9050 '
9060 IF ERR > 4 OR ERR < 1 GOTO 9240        ' Erreur non prevue
9070 '
9080 ' La valeur de ERR permet de selectionner le message correspondant
9100 '
9110 ' En regle generale, le message est contenu dans D$(ERR)
9120 '
9130 A$ = D$(ERR)           ' Transfere le diagnostic en A$
9140 A$ = A$+" "            ' Ajoute un espace pour separer du prochain mot
9150 '
9160 ' La valeur de ORG est utilisee de maniere analogue a ERR
9170 ' pour selectionner la description du sous-programme
9180 '
9190 IF ORG > 3 OR ORG < 1 GOTO 9250        ' Provenance non prevue
9200 A$ = A$+R$(ORG)                        ' Ajoute a la chaine A$ la provenance
9210 LPRINT A$                              ' Ecrit le diagnostic en entier
9220 ERR = 0 : ORG = 0
9230 RETURN
9240 LPRINT "Erreur non prevue"
9250 LPRINT "Provenance non prevue"
```

```
Erreur n. : 1
Provenance : 1
Erreur de lecture ou d'ecriture sur disque - Premier sous-programme
Erreur n. : 1
Provenance : 2
Erreur de lecture ou d'ecriture sur disque - Calcul interets
Erreur n. : 1
Provenance : 3
Erreur de lecture ou d'ecriture sur disque - Sous-programme d'essai

Erreur n. : 2
Provenance : 1
Saisie de donnees non valides - Premier sous-programme
Erreur n. : 2
Provenance : 2
Saisie de donnees non valides - Calcul interets
Erreur n. : 2
Provenance : 3
Saisie de donnees non valides - Sous-programme d'essai
```

```
Erreur n. : 3
Provenance : 1
Choix d'une option non prevue - Premier sous-programme
Erreur n. : 3
Provenance : 2
Choix d'une option non prevue - Calcul interets
Erreur n. : 3
Provenance : 3
Choix d'une option non prevue - Sous-programme d'essai

Erreur n. : 4
Provenance : 1
Choix d'une option non prevue - Premier sous-programme
Erreur n. : 4
Provenance : 2
Choix d'une option non prevue - Calcul interets
Erreur n. : 4
Provenance : 3
Choix d'une option non prevue - Sous-programme d'essai

Erreur n. : 9
Provenance : 1
Erreur non prevue
```

Exécution de calculs avec un nombre quelconque de chiffres significatifs. Dans le cadre des applications pratiques courantes, l'exactitude des calculs obtenus avec des nombres en double précision est généralement satisfaisante.

Toutefois, il s'avère parfois nécessaire d'exécuter les calculs avec une meilleure précision. Différentes méthodes permettent de résoudre ce problème, mais toutes impliquent des difficultés de programmation parfois considérables.

La question de la précision des résultats doit être envisagée selon deux points de vue différents : d'une part le domaine scientifique, et d'autre part les problèmes de calculs économiques.

Dans le cadre des applications scientifiques, on ne devrait pas rencontrer de difficultés liées à la précision des calculs : on décrit des phénomènes physiques réels (et donc mesurables), et l'on connaît d'avance l'ordre de grandeur des variables qui les représentent. De plus, on sait que les dernières décimales d'un résultat n'ont absolument aucune signification physique.

Ainsi, lorsque l'exécution d'un calcul demande l'utilisation de précisions trop élevées, on doit en rechercher la cause dans la conception même du programme (algorithme), ou dans une mauvaise utilisation, et non dans une trop faible précision de la machine.

Dans le domaine économique, par contre, on raisonnera de façon opposée : pour présenter un bilan, ou tout autre document comptable, les résultats doivent apparaître jusqu'à la dernière décimale, quelque soit la grandeur du nombre (tous les chiffres sont significatifs). Prenons un exemple : d'un point de vue physique, estimer la vitesse d'une voiture à 100 km/heure, au lieu de 100.02 km/heure, se justifie aisément ; en revanche, si le nombre représentait le montant d'une facture participant au calcul d'un bilan, l'arrondi provoquerait une erreur dans la vérification définitive des comptes.

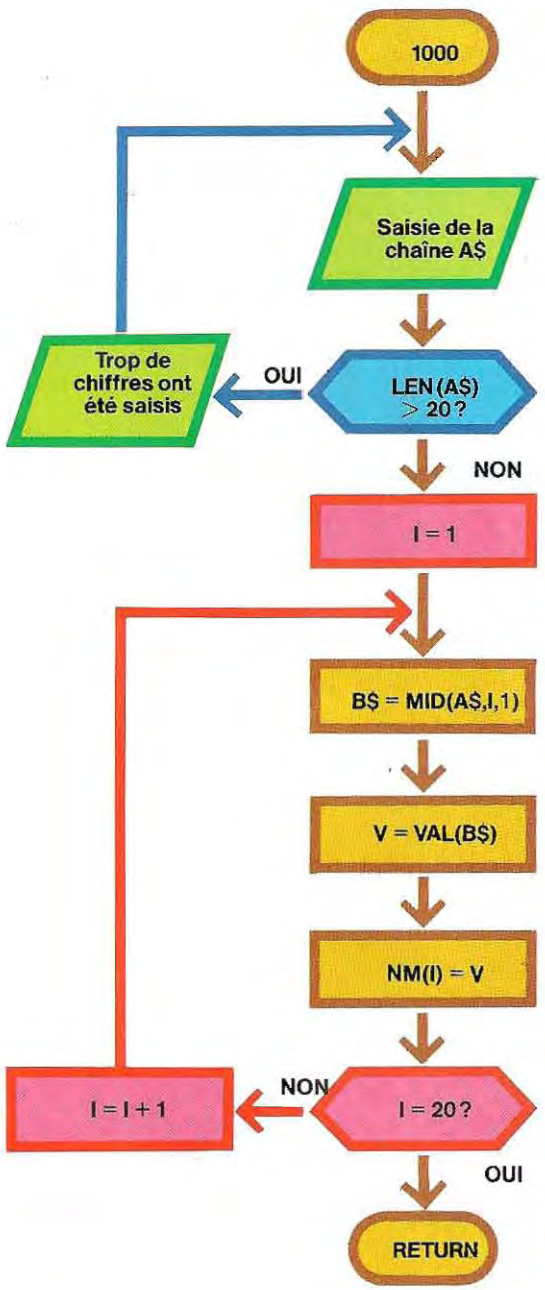
Une méthode simple pour se dégager des limitations de la machine (18 chiffres significatifs au maximum sur les plus gros systèmes), revient à considérer chaque nombre comme une chaîne de caractères. De cette manière, le nombre maximal de chiffres qu'il est possible de traiter augmente, selon les systèmes, jusqu'à 128 ou 256 (longueur maximale d'une chaîne).

Examinons maintenant une méthode permettant d'additionner deux nombres de longueur quelconque, saisis et représentés en mémoire sous forme de chaînes. Si l'on voulait entrer dans l'ordinateur un nombre de 20 chiffres en format numérique, celui-ci serait tronqué pour respecter la précision de la machine.

Ce même nombre, lu en tant que variable

alphanumérique, sera directement et entièrement saisi, comme une quelconque chaîne de 20 caractères.
On extraira successivement chaque caractère ASCII de la chaîne, on le transformera en valeur numérique (instruction VAL), pour le stocker ensuite dans un tableau.
De cette manière, la représentation numérique en mémoire n'est plus une valeur décimale (un nombre unique), mais un ensemble de chiffres séparés, chacun dans une case mémoire distincte. En machine, les problèmes

SOUS-PROGRAMME DE SAISIE D'UN NOMBRE EN « PRECISION ETENDUE »



Boucle de conversion des chiffres

Fonctions E/S

Contrôle des saisies

La chaîne A$ contient le nombre sous forme de caractères.

Ce contrôle est nécessaire pour ne pas déborder du tableau qui devra contenir les valeurs numériques. Pour accroître la précision, il suffit d'augmenter les dimensions du tableau NM.

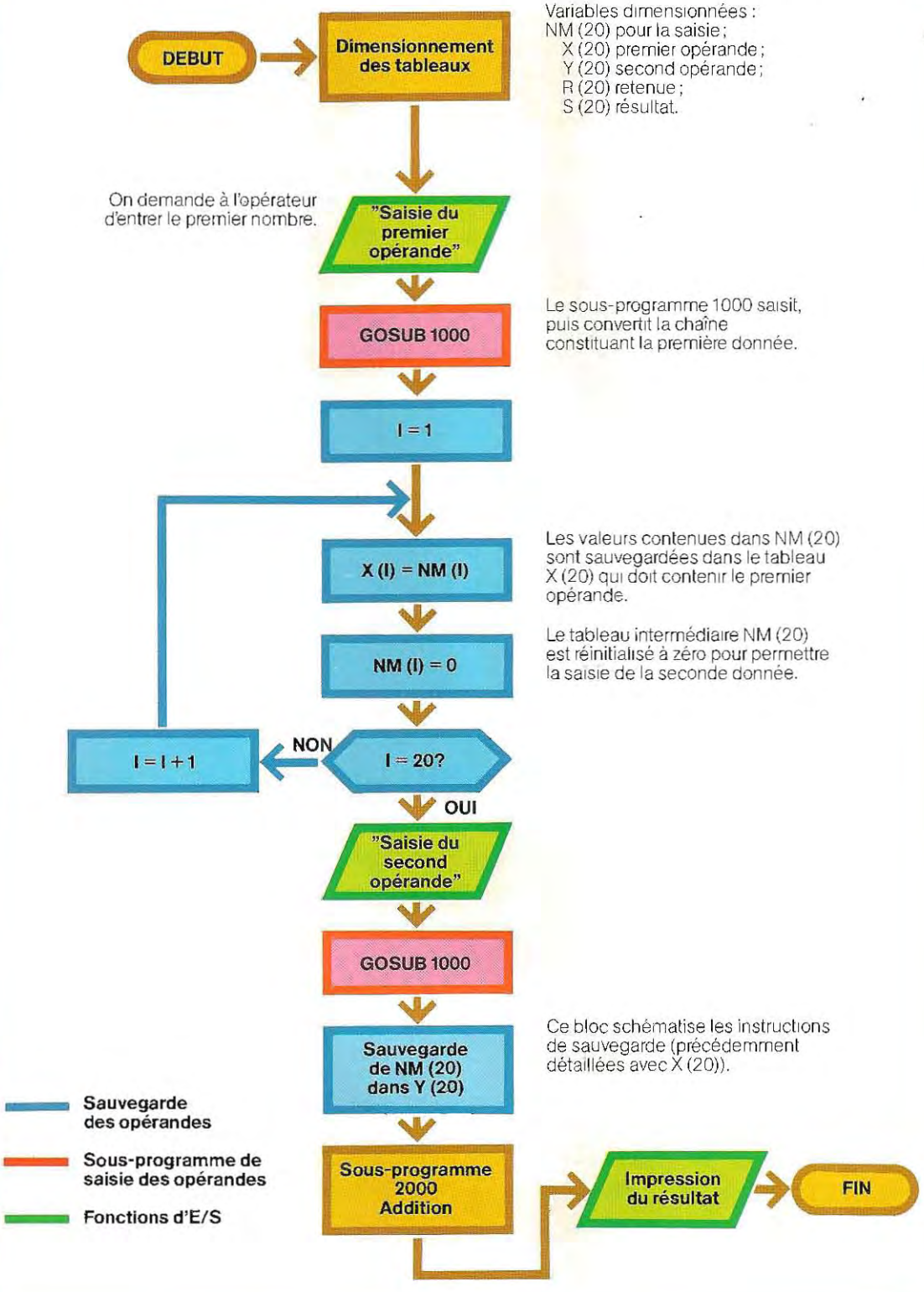
Extrait de A$ le caractère en position I.

Le convertit en valeur numérique.

Chaque chiffre du nombre initial est mémorisé dans un élément du tableau NM (20).

En sortie du sous-programme, le tableau NM (20) contient le nombre saisi (un chiffre par élément).

ADDITION DE DONNEES NUMERIQUES EN « PRECISION ETENDUE »



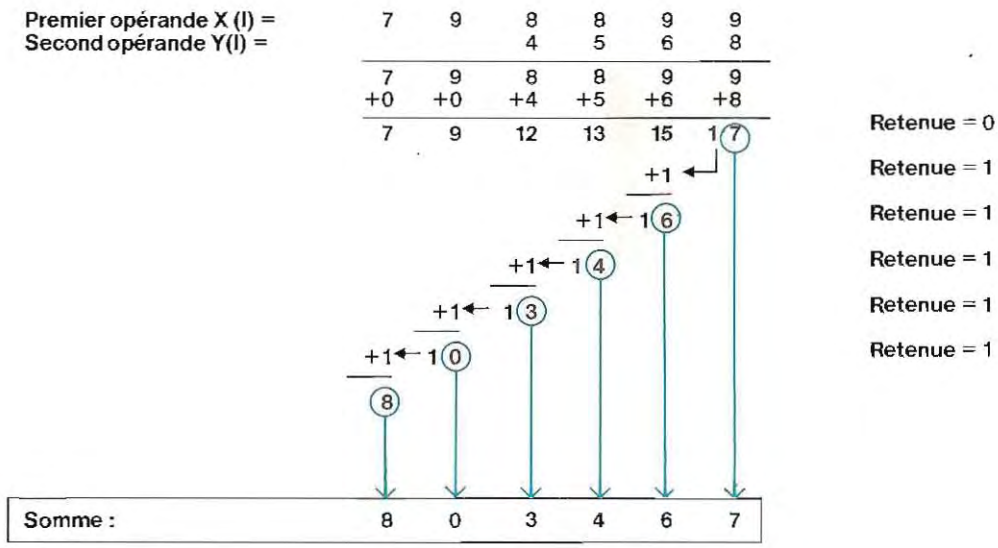
de précision disparaissent : le nombre est représenté par un tableau, chaque élément contenant un seul chiffre compris entre 0 et 9. L'organigramme page 532 retrace le déroulement du sous-programme 1000, qui saisit, sous forme de chaîne, un nombre allant jusqu'à 20 chiffres, puis convertit chaque caractère en valeur numérique, pour le stocker dans le tableau intermédiaire NM.

Page 533, est représenté l'organigramme du programme principal, contenant deux appels successifs au sous-programme précédent, puis un appel à la routine d'addition. Le calcul de la somme, élément par élément, s'effectue

EXEMPLE DE CALCUL UTILISANT UN TABLEAU DE CHAINES

```
10 ' FICHIER = ST1
15 DEFINT A-Z
20 DIM NM(20),X(20),Y(20),R(20),S(20)
30 PRINT "Saisie du premier operande"
40 GOSUB 1000
45 LPRINT " VALEUR PREMIER OPERANDE = ";A$
50 FOR I=1 TO 20
60 X(I)=NM(I)
70 NM(I)=0
80 NEXT I
90 PRINT "SAISIE DU SECOND OPERANDE"
100 GOSUB 1000
105 LPRINT " VALEUR SECOND OPERANDE = ";A$
110 FOR I=1 TO 20
120 Y(I)=NM(I)
130 NM(I)=0
140 NEXT I
150 GOSUB 2000
160 PRINT "RESULTAT:" : LPRINT : LPRINT " RESULTAT : "
170 FOR I=1 TO 20
175 PRINT S(I);        ' Le resultat sort a la fois
176 LPRINT S(I);       ' sur l'ecran et l'imprimante
177 NEXT I
180 END
1000 ' ** SOUS-PROGRAMME DE SAISIE **
1010 INPUT "Entrer la valeur";A$
1020 IF LEN(A$)>20 THEN PRINT "Erreur":GOTO 1010
1030 FOR I=1 TO 20
1040 B$=MID$(A$,I,1)
1050 V=VAL(B$)
1060 NM(I)=V
1070 NEXT I
1080 RETURN
2000 ' ** SOUS-PROGRAMME DE CALCUL **
2010 FOR I=20 TO 1 STEP-1
2020 V=X(I)+Y(I)              ' Calculer la somme des deux elements en position I
2030 IF I=20 GOTO 2050        ' Pas de retenue au premier passage
2040 V=V+R(I+1)               ' Ajouter la retenue de l'addition precedente
2050 V1=INT(V/10)
2060 R(I)=V1                  ' Memoriser la nouvelle retenue, qui sera ajoutee
2065 '                        ' au prochain passage
2070 V2=V-V1*10               ' Isoler les unites, qui constituent le chiffre
2075 '                        ' en position I du resultat
2080 S(I)=V2                  ' Memoriser ce chiffre dans le tableau resultat
2090 NEXT I
2095 RETURN

 VALEUR PREMIER OPERANDE = 01234567899876543210
 VALEUR SECOND  OPERANDE = 09876543210123456789

 RESULTAT :
 1  1  1  1  1  1  1  1  1  0  9  9  9  9  9  9  9  9  9  9

 VALEUR PREMIER OPERANDE = 01234567899876543210
 VALEUR SECOND  OPERANDE = 89764532136987412884

 RESULTAT :
 9  0  9  9  9  1  5  0  0  3  6  8  6  9  9  5  5  5  1  4
```

ADDITION DE DEUX ENTIERS EN « PRECISION ETENDUE »



Test 15

1 / Parmi les suivantes, certaines définitions de fonctions sont erronées : lesquelles et pourquoi ?

a) DEF FNA=CHR$(27)+CHR$(10)
b) DEF FNB=C*2+1
c) DEF FNN$=CHR$(7)+"Valeur"
d) DEF FNA$="Chaîne d'essai"+"="+256

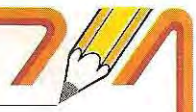
2 / Ecrire un programme réalisant les fonctions suivantes :

- saisir une chaîne et une valeur numérique comprise entre 0 et 99 ;
- vérifier que la valeur numérique saisie représente un caractère alphabétique en ASCII ;
- dans l'affirmative, rechercher le rang de la première apparition de ce caractère dans la chaîne ;
- stopper l'exécution lorsque l'opérateur entre la chaîne "FIN".

3 / Ecrire un programme permettant de :

a) lire une suite de données entrées au clavier, chacune étant constituée par un groupe de nombres :

code = valeur numérique, comprise entre 10 et 20 (11 codes possibles),
coût unitaire = valeur numérique, comprise entre 1250 et 5700,
quantité = valeur numérique quelconque,
nom = chaîne alphanumérique de 20 caractères.



La saisie doit se terminer lorsque le Code lu est nul ;
b) vérifier que le Nom saisi figure parmi 10 noms préétablis dans un Data ;
c) calculer le coût total pour chaque donnée (coût total = Coût unitaire × Quantité) ;
d) effectuer la somme des Quantités saisies et des coûts totaux (calculés précédemment) pour chaque Code ;
e) calculer la somme des coûts totaux pour chaque Nom ;
f) afficher les sommes calculées lors des deux étapes précédentes. Lorsque le Code lu est nul, sortir de la boucle de saisie. Il est conseillé d'établir le programme en définissant les tableaux suivants :
CODE(11) pour les 11 valeurs des totaux par code,
NOM$(10) chaînes contenant les 10 noms prévus,
NMT(10) totaux par nom.

---

4 / Ecrire un sous-programme s'insérant dans le programme précédent, pour calculer le prix unitaire moyen, toutes données confondues. On se souviendra que la moyenne de N valeurs est donnée par leur somme divisée par N.

---

5 / La factorielle d'un nombre entier N est notée N! ; elle est obtenue en multipliant entre eux tous les nombres entiers compris entre 1 et N :
Factorielle = N! = $1 \times 2 \times 3 \ldots \times (N-1) \times N$
Par exemple :
$5! = 1 \times 2 \times 3 \times 4 \times 5 = 120 \quad 1! = 1$
Lorsque N est nul, on adopte la convention mathématique suivante : 0! = 1.
Ecrire un sous-programme calculant la factorielle d'un nombre entier NUM.

---

6 / Les « combinaisons de N éléments pris K à K » représentent toutes les manières possibles de choisir un groupe de K éléments parmi N, et sans répétition (un élément ne peut apparaître plus d'une fois dans le même groupe).
Par exemple, en disposant des lettres A, B et C (N = 3), et en extrayant les lettres deux à deux (K = 2), on peut obtenir les groupes : AB, AC et BC.
D'une façon générale, le nombre de choix possibles (nombre de combinaisons) est donné par l'expression :

$$\begin{pmatrix}\text{nombre de combinaisons} \\ \text{de N éléments pris K à K}\end{pmatrix} = \frac{N!}{K! \times (N-K)!}$$

On note cette valeur $\binom{N}{K}$ ou C_N^K

Dans l'exemple précédent (N = 3, K = 2), on a :

$$\text{nombre de combinaisons} = \frac{3!}{2! \times (3-2)!} = \frac{1 \times 2 \times 3}{1 \times 2 \times 1} = 3$$

Ecrire un programme calculant le nombre de combinaisons possibles de K éléments choisis parmi N.

Les solutions du test se trouvent pages 545, 546 et 547.

dans le sous-programme 2000, dont le listing est illustré page 534. La page 535 explicite la logique suivie : l'exemple détaille l'addition de deux nombres entiers (798899 et 4568), selon la méthode utilisée dans le sous-programme 2000. Remarquons que la première donnée (798899) « déborderait » largement du format entier de la machine (16 bits correspondent à l'intervalle [– 32768 ; + 32767]).

Applications à la statistique. Pour décrire des phénomènes de masse tels que la répartition de la population dans un pays déterminé ou les ventes moyennes annuelles par client dans une société, il est nécessaire de disposer d'un grand nombre d'observations individuelles*, supposées représentatives, qui permettront ensuite d'extrapoler les résultats partiels à l'ensemble étudié.
Il serait impossible d'obtenir manuellement une si grande quantité de données. Seul l'ordinateur peut satisfaire les exigences (précision et rapidité) du calcul statistique. C'est donc l'instrument indispensable à l'étude d'un phénomène : des programmes très généraux, acceptant en entrée un nombre quelconque d'observations de types très divers, exécuteront les calculs nécessaires pour obtenir les données statistiques décrivant le phénomène.
A titre d'exemple, sont développés ici certains programmes de calcul des grandeurs statistiques.
Supposons que nous voulions effectuer la synthèse d'un ensemble de données observées. On cherchera à déterminer une caractéristique globale de notre « échantillon », un paramètre qui, substitué aux différentes valeurs observées, n'en altère pas les caractéristiques statistiques. Cette grandeur, que nous appellerons une moyenne, permet des évaluations globales rapides, facilite les comparaisons entre échantillons, et sert de guide dans les situations incertaines.
On distingue des types différents de moyenne, selon la fonction caractéristique des données que l'on veut observer (moyenne

* Le résultat d'une observation individuelle est appelé **unité statistique** ; le résultat d'une opération réalisée sur des unités statistiques est appelé **donnée statistique.**

L'ordinateur est l'instrument indispensable de toute étude statistique. Ici, nous voyons (au fond de la salle et sur le chariot) les tambours de disques permettant le stockage des données.

ISTAT

géométrique, moyennes harmonique, médiane, etc.). La plus fréquente en calcul statistique est la **moyenne arithmétique,** que l'on calcule de la manière suivante :

$$\text{Moyenne arithmétique} = \frac{\text{somme des données}}{\text{nombre de données}}$$

Elle s'exprime toujours dans l'unité de mesure des données prises en compte par le calcul, et se représente mathématiquement par la variable $\overline{X}$.
Pour évaluer la moyenne des revenus annuels de six familles, soit :

$X_1 = 104\,500$ F/an
$X_2 = 109\,500$ F/an
$X_3 = 123\,000$ F/an
$X_4 = 151\,000$ F/an
$X_5 = 176\,500$ F/an
$X_6 = 186\,500$ F/an

on calculera d'abord le revenu annuel total, égal à la somme des six revenus :

$X_1 + X_2 + X_3 + X_4 + X_5 + X_6 = 851\,000.$

Le revenu moyen vaudra :

$\overline{X} = 851\,000 : 6 = 141\,833.33$ F/an.

Ce revenu constituerait celui de chaque famille, si le revenu total de 851 000 F était réparti de façon égale entre les six familles.
Le programme de la page 539 calcule la moyenne arithmétique d'un certain nombre de données (MAX) entrées au clavier. Il utilise un tableau dimensionné à 100 éléments : il pourra, au maximum, calculer la moyenne de 100 données.
Les différents types de moyenne sont obtenus en combinant les données observées de plusieurs façons. Il existe d'ailleurs une moyenne qui ne prend en compte que la position des données. Il s'agit de la **valeur médiane** (ou moyenne probable), qui n'est autre que la moyenne arithmétique des deux termes au centre de la série ordonnée des n valeurs (quand n est pair) ou le terme central (quand n est impair).
Un exemple simple démontrera qu'il est parfois préférable de calculer la médiane d'un ensemble d'observations plutôt que sa moyenne. On doit approvisionner en pain cinq magasins d'alimentation (A, B, C, D et E), chacun situé dans différents quartiers de la ville, mais tous près d'une route qui la traverse. On suppose donc que les magasins se trouvent sur la route, répartis aux kilométrages suivants :

magasin	A	B	C	D	E
distance en km	1	2	3	8	11

On cherche, pour construire un four, le point le plus près possible des cinq magasins. Le meilleur endroit sera celui qui, en additionnant les distances absolues entre le four et chaque magasin, donnera le résultat minimum, et ce lieu sera la valeur de la médiane. En effet, celle-ci correspond à l'élément central (3) et donne une somme de distances égale à 16 km ; en considérant la moyenne arithmétique (5), on obtient 18 km.
Introduisons maintenant une nouvelle grandeur statistique. Il peut s'avérer intéressant de connaître la fluctuation des données autour de la valeur moyenne (on veut, par exemple, savoir si les prix réels observés approchent plus ou moins le prix moyen). L'amplitude de cette variation indique si la moyenne est bien représentative de l'échantillon observé.
On calcule un nouveau paramètre statistique appelé variance, et noté mathématiquement par le symbole σ^2, de la manière suivante :

$$\sigma^2 = \frac{\text{somme des carrés des écarts}}{\text{nombre de données}}$$

où le terme « écart » représente l'écart à la moyenne (différence entre la valeur de chaque donnée et la moyenne). La variance peut aussi s'exprimer sous la forme :

$$\sigma^2 = \frac{\text{somme des carrés des données}}{\text{nombre de données}} - \text{carré de la moyenne}$$

Pour illustrer ces formules, considérons, par exemple, les prix de différentes qualités d'un certain produit :

Qualité	Prix (francs/kg)
A	$X_1 = 2.10$
B	$X_2 = 3.40$
C	$X_3 = 4.50$
D	$X_4 = 5.10$
E	$X_5 = 5.50$
F	$X_6 = 6.00$
G	$X_7 = 6.30$

PROGRAMME DE CALCUL DES MOYENNES

```
10 ' ** PROGRAMME DE CALCUL DES MOYENNES
20 '
30 '   FICHIER : STAT 1
35 B$="£££.£"          ' Format d'impression
40 '
45 DEFDBL A-H
46 DEFINT I-N
47 DEFDBL O-Z
50 DIM A(100)          ' Tableau memorisant les observations
60 '                    au maximum 100 valeurs
70 INPUT "Sur combien de donnees portera le calcul ? ";MAX
80 IF MAX=0 OR MAX>100 THEN PRINT " ** ERREUR ** ":GOTO 70
90 ' * Saisie des valeurs
100 FOR I=1 TO MAX
110 PRINT " Donnee num. :";I
120 INPUT "Valeur      :";A(I)
130 NEXT I
140 ' ** CALCUL DU TOTAL **
150 T=0                        ' Initialisation a zero de la somme
160 FOR I=1 TO MAX
170 T=T+A(I)                   ' Accumulation dans T de la somme des donnees
180 NEXT I
190 ' ** Les lignes 160 a 180 peuvent etre eliminees
200 '    en calculant simplement la somme pendant la saisie
210 '    c.a.d. en ecrivant en ligne 125 l'instruction T=T+A(I)
220 '
230 TMOY=T/MAX                 ' Le nom utilise pour la moyenne doit commencer
231 '                           par une des lettres reservees a la double
232 '                           precision
233 '
234 '
240 '
250 ' ** Impression
260 '
270 LPRINT "Donnees :"
280 ' Les donnees sont imprimees par lignes de 10
290 '
300 FOR I=1 TO MAX STEP 10
305 I1=I+9
310 FOR J=I TO I1
320 LPRINT USING B$;A(J);
330 NEXT J
340 LPRINT
350 NEXT I
360 '
365 LPRINT:LPRINT
370 LPRINT " Moyenne : ";TMOY
380 '
390 INPUT " Voulez-vous continuer (OUI/NON) :";REP$
400 IF REP$="OUI" GOTO 70
410 END
```

```
Donnees :
10.2  15.0  15.2  14.4  13.2  10.0  11.0  10.9  11.6  12.0
12.8  12.4  15.9  16.7  16.1  16.8  11.2  10.0  12.0  13.0
14.0  15.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0   0.0

Moyenne : 13.1554545454545
```

La somme des données est de 32.90 F.
La moyenne des prix vaut

$$32.90 : 7 = 4.70$$

Pour évaluer la variance (avec la seconde formule), on calcule :

$$X_1^2 = \quad 4.41$$

La compréhension de la parole

D'une certaine manière, un ordinateur est analogue à un cerveau : tous deux assimilent des informations en provenance d'organes sensoriels externes, et tous deux les restituent, après traitement, à d'autres organes externes. Cependant, on constate une différence radicale : le cerveau possède une aptitude supérieure à l'ordinateur pour appréhender les informations, mais en revanche, ce dernier le dépasse en vitesse quand il s'agit de traiter des données numériques. Ceci s'explique probablement par le fait que la saisie des informations par le cerveau s'effectue en parallèle et d'une manière fortement redondante, alors qu'un ordinateur acquiert ses données séquentiellement.

La ***reconnaissance de la parole*** *est à la fois un problème de reconnaissance des formes et d'intelligence artificielle. Son intérêt est évident, la parole constituant le moyen le plus rapide et le plus naturel d'accéder à l'information. Elle jouera certainement un rôle de plus en plus important dans la communication entre l'homme et la machine. Mais le jour où les ordinateurs reconnaîtront et comprendront la parole humaine n'est pas encore arrivé, et cela en raison de multiples problèmes que l'on connaît aujourd'hui, et que l'on s'attache à résoudre.*

En effet, pour comprendre la parole, un homme ne considère pas seulement l'information perçue par l'oreille, mais aussi le ***contexte*** *dans lequel elle a été émise. C'est pour cela que nous comprenons le langage parlé, même lorsque le signal de parole est entaché de bruit. Cela dit, comprendre le contexte du discours suppose une vaste connaissance du monde environnant. Et c'est ici que la reconnaissance de la parole échoue aujourd'hui.*

Il est difficile de réaliser des programmes informatiques suffisamment sophistiqués pour comprendre la parole ***continue*** *(naturelle), d'un locuteur a priori inconnu de la machine. La reconnaissance de la parole par l'ordinateur est possible quand on simplifie le problème : en prononçant les mots séparés par des silences (c'est la reconnaissance de* ***mots isolés****), en limitant la taille du vocabulaire de référence, en restreignant le nombre de locuteurs qui devront être reconnus.*

On peut déjà acheter des jouets et des jeux vidéo qui obéissent à la voix pour moins de 1 000 F. Le prix des produits commercialisés de reconnaissance vocale varie entre 4 000 F et 400 000 F, suivant leurs performances et la taille du vocabulaire traité. La société japonaise Matsushita a présenté au « 1983 Japan Data Show » un module de reconnaissance vocale portable, avec clavier incorporé, affichage sur 20 caractères et haut-parleur intégré, qui reconnaît 62 mots isolés. La société Crouzet utilise un module de reconnaissance de mots enchaînés, en collaboration avec le LIMSI (Laboratoire d'Informatique pour la Mécanique et les Sciences de l'Ingénieur, CNRS). Ces produits imposent des contraintes à l'utilisateur : le vocabulaire est limité (de 10 à 200 mots), une phase d'apprentissage est nécessaire pour chaque nouveau locuteur, et les mots doivent être prononcés distinctement.

Quels sont les problèmes rencontrés en reconnaissance automatique de la parole ?

Le signal de parole peut être segmenté en une suite de sons élémentaires appelés ***phonèmes****, qui présentent des caractéristiques articulatoires et acoustiques particulières. Bien que l'appareil vocal humain puisse produire un nombre presque infini de positions articulatoires, le nombre de phonèmes reste limité : 36 en français, 22 en espagnol, 44 en anglais. Chaque phonème possède des caractéristiques acoustiques distinctes. Combiné avec d'autres phonèmes, il forme des unités phonétiques plus grandes, telles que les diphonèmes, les demi-syllabes, les syllabes. Une connaissance approfondie des différences acoustiques entre ces unités sonores est nécessaire à la distinction de mots tels que : « cabot », « canot », « cadeau », « cachot »...*

Quand les sons élémentaires sont assemblés pour former des unités phonétiques plus grandes, les caractéristiques acoustiques d'un phonème donné seront modifiées par le phonème qui le précède et celui qui le suit ; cela s'explique par l'interaction physique entre les différentes structures anatomiques (la langue, les lèvres, les cordes vocales, le nez, etc.). Le même phonème de base, « t » par exemple, aura des caractéristiques totale-

ment différentes dans des mots (ou contextes) différents tels que «thé», «trait», «temps», «battu», «astre»... Cet effet de **coarticulation** nous rend particulièrement délicate la reconnaissance d'un phonème à l'intérieur d'un mot.

Une autre difficulté est la **continuité** du langage parlé. En effet, les mots sont séparés dans un texte écrit, alors qu'ils ne le sont pas dans un texte parlé. Au départ, pour l'ordinateur, la parole est uniquement une suite continue de sons. Ainsi, si la phrase prononcée est «J'ai mal au pied», et si l'ordinateur se limite à des connaissances phonétiques, il ne saura pas faire la différence entre «J'aima l'haut pied» et «géai mâle au pied». Dans le cas de cette phrase précise, plus de mille combinaisons sont possibles. L'ordinateur a besoin d'autres connaissances.

On va alors le doter de **connaissances syntaxiques** (règles de grammaire), qui lui permettront d'éliminer les phrases de structure syntaxique incorrecte. Le nombre de combinaisons possibles au niveau syntaxique sera plus faible, mais conservera des phrases qui n'auront aucun sens, comme: «j'aima l'haut pied» (sujet - verbe - article - adjectif - complément).

Pour éliminer les combinaisons qui n'ont pas de sens, on fait appel à des **connaissances sémantiques** qui sont très difficiles à formaliser en langage informatique. Ainsi, après s'être référé successivement aux connaissances phonétiques, lexicales (le dictionnaire de la langue), syntaxiques et sémantiques, deux combinaisons restent candidates: «J'ai mal au pied» et «J'ai mal aux pieds». Et là, seules des considérations sur le contexte pourront nous indiquer la phrase à retenir. Si, dans la phrase précédente, il s'agissait du pied droit, on conservera «J'ai mal au pied». Dans ce cas, on a fait appel à des connaissances au niveau **pragmatique** (lié au contexte).

Un système de reconnaissance de la parole continue doit rassembler une grande quantité de connaissances dans des domaines extrêmement variés: acoustiques (sur le signal de parole), phonétiques (sur la langue concernée), lexicales (dictionnaire de la langue), syntaxiques (structure des phrases), sémantiques (sens des mots), pragmatiques (contexte des mots). Les futurs ordinateurs devront donc accéder à beaucoup de connaissances.

De plus, il faut pouvoir raisonner sur ces connaissances, et de manière astucieuse et non exhaustive: c'est le rôle des stratégies de raisonnement. Celles-ci prennent une place importante dans les systèmes de reconnaissance de la parole continue actuels. En effet, quelles connaissances utiliser, et dans quel ordre?

On distingue, par exemple, les **stratégies ascendantes** (utilisant les connaissances dans l'ordre suivant: acoustique - phonétique - lexical - syntaxique - sémantique - pragmatique), comme dans l'exemple précédent, et des **stratégies descendantes**, dans lesquelles on part d'hypothèses issues du contexte pour redescendre l'échelle des niveaux de compréhension: pragmatique - sémantique - syntaxique - lexical - phonétique - acoustique. Ainsi, dans l'approche descendante, on va émettre des hypothèses, faires de prédictions sur ce qui va être dit, puis les vérifier.

Faut-il utiliser une stratégie ascendante ou descendante? Il semble que la première soit plus performante quand on n'a pas commis trop d'erreurs de reconnaissance phonétique, et que l'univers d'application (lexique, syntaxe, sémantique) est large. Par contre, quand ce dernier est restreint (c'est-à-dire que le contexte est bien connu) et qu'il y a trop d'erreurs de reconnaissance phonétique, la stratégie descendante s'avère plus performante. En fait, les systèmes actuels comportent des stratégies mixtes.

On voit donc apparaître une grande variété de stratégies de reconnaissance, quelquefois très complexes, exploitant des connaissances à des niveaux très différents, et les parcourant de multiples façons selon la phrase à reconnaître.

Où en sommes-nous aujourd'hui?

Aux Etats-Unis, la recherche en reconnaissance de la parole continue fut largement encouragée et coordonnée par le projet ARPA (1971-1976), financé par le Département de la Défense américain. Depuis, elle s'est assoupie, exception faite de l'effort considérable d'IBM pour réaliser une machine à écrire à entrée vocale.

En France, des systèmes de reconnaissance de la parole continue existent au LIMSI (sys-

Phrase orthographique :	J'AI MAL AU PIED
Phrase phonétique :	JÉ MAL O PYÉ
Phrases possibles au niveau lexical :	J'AIMA L'HAUT PIED GEAI MÂLE AU PIED JET MALLE EAU PIEDS ... (PLUS DE 1000 COMBINAISONS POSSIBLES)
Phrases possibles au niveau syntaxique :	J'AIMA L'HAUT PIED J'AI MAL AU PIED GEAI MÂLE AU PIED J'AI MAL AUX PIEDS
Phrases possibles au niveau sémantique :	J'AI MAL AU PIED J'AI MAL AUX PIEDS
Phrase possible au niveau pragmatique :	J'AI MAL AU PIED

Exemple de stratégie ascendante en reconnaissance de la parole continue (d'après J. MARIANI).

tème ESOPE), au CNET (système KEAL), au CRIN (MYRTILLE), au CERFIA (ARIAL), à l'ENSERG, à Marseille-Lumigny et au LEA de Nancy.

Il est intéressant de noter que la machine à écrire à entrée vocale fait partie des applications pilotes envisagées par les Japonais dans leur projet d'ordinateur de la cinquième génération. Mais les Etats-Unis et les pays européens ne doivent pas être effrayés, car la langue japonaise est beaucoup plus simple à reconnaître que les autres. On peut d'ailleurs s'étonner de la modestie de leurs ambitions, étant donné les progrès technologiques considérables que suppose l'aboutissement de leur projet.

Ce qui reste intéressant dans ce projet, c'est l'accent mis sur la réalisation de machines spécialisées dans le traitement des connaissances. L'informatique classique s'appuie sur une structure de contrôle, des ***programmes*** *et des* ***données****. L'évolution récente vers l'intelligence artificielle se fonde au contraire sur la qualité entre les* ***inférences*** *(le raisonnement) et les* ***connaissances****.*

C'est ce que l'on constate déjà dans les techniques de ***« systèmes experts »*** *qui commencent à voir le jour au niveau commercial. Ce type de système est capable de raisonner à partir de connaissances de haut niveau fournies par des experts humains. Déjà, des systèmes experts en diagnostic médical dépassent les performances des spécialistes, puisqu'ils réalisent la synthèse des connaissances acquises dans ce domaine particulier. Leur rôle ne consiste pas à supplanter l'homme, mais plutôt à l'aider dans ses décisions, à lui apprendre et lui expliquer ce que d'autres hommes savent. C'est ainsi que déjà plusieurs sociétés se proposent de construire et de commercialiser des systèmes experts. Mais, à l'heure actuelle, ils ne s'avèrent capables de raisonner que dans des domaines d'expertise très restreints. On entendra parler d'un* ***système expert en*** *prospection minière (PROSPECTOR),* ***en*** *diagnostic d'affections bactériennes et indications thérapeutiques (MYCIN), etc., et non d'un* ***système expert.***

Cette approche a beaucoup séduit les cher-

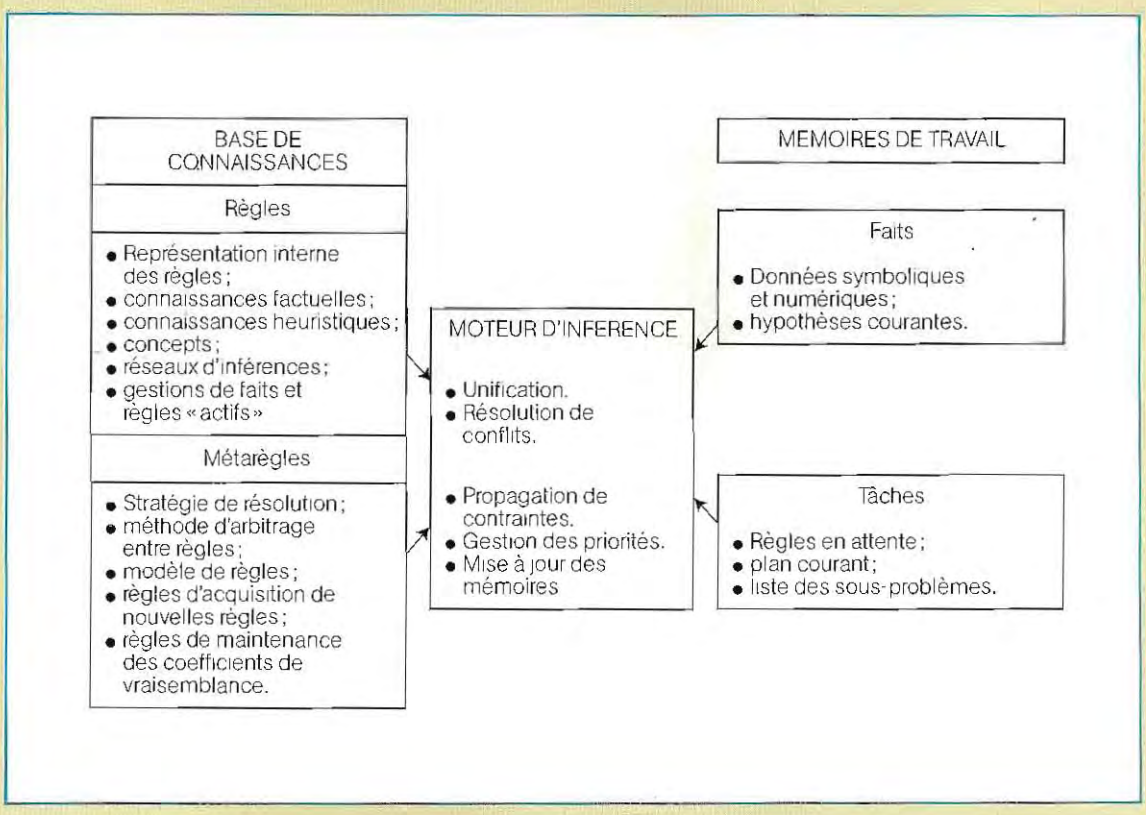


Schéma général d'un système expert.

cheurs en reconnaissance de la parole continue. Les ordinateurs de la cinquième génération y seront d'ailleurs parfaitement adaptés. C'est ainsi qu'aux Etats-Unis, A. Newell et D. McCraken ont récrit les systèmes HEARSAY et HARPY, réalisés à la Carnegie-Mellon University au cours du projet ARPA. De même, les laboratoires de la CGE à Marcoussis ont réalisé une version similaire d'HEARSAY, et le CNET, à Lannion, a récrit son système KEAL sous forme de système expert.

Une orientation récente de quelques laboratoires consiste à réaliser des systèmes experts capables de reconnaître des ***sonagrammes*** *(images d'une émission vocale dans les coordonnées temps - fréquence - amplitude). En effet, un chercheur américain du MIT (V. Zue) arrive à lire des sonagrammes provenant de plusieurs locuteurs, avec moins de 10 % d'erreurs. Ainsi, la réalisation d'un système expert en lecture de sonagrammes devrait permettre d'obtenir des connaissances phonétiques plus fondamentales et mieux exploitables.*

Cette idée est en cours de réalisation en France (CRIN, LIMSI), en Angleterre (Université de Cambridge), et aux Etats-Unis (MIT, VERBEX). Il est clair qu'un tel système serait susceptible d'intéresser au plus haut point les laboratoires qui n'ont pas d'expert en sonagrammes à leur disposition.

Enfin, le groupe d'intelligence artificielle de l'université de Lumigny-Aix-Marseille poursuit des études en traitement automatique de la parole.

Ainsi, il est clair que l'intelligence artificielle, sous ses divers aspects (résolution de problèmes, reconnaissance de la parole continue, compréhension du langage naturel, etc.) permettra à l'homme d'utiliser avec souplesse des systèmes informatiques de plus en plus performants, destinés à jouer un rôle croissant dans sa vie.

• « Les Ordinateurs de la cinquième génération », LA RECHERCHE, n° 154, avril 1984.
• « ESOPE : un système de compréhension de la parole continue », thèse d'Etat, Joseph MARIANI, Paris VI, 1982.
• « Artificial Intelligence », IEEE-SPECTRUM, novembre 1983.

$X_2^2 = 11.56$
$X_3^2 = 20.25$
$X_4^2 = 26.01$
$X_5^2 = 30.25$
$X_6^2 = 36$
$X_7^2 = 39.69$

La somme vaut 132.53 et donc :

$$\frac{\text{somme des carrés des données}}{\text{nombre de données}} - (\text{moyenne})^2 =$$

$= 132.53 : 7 = 22.09 = 18.932857 - 22.09 =$
$= 3.157143$

Comme indice de variation, on considère la racine carrée de la variance. Cette valeur, désignée par σ, est appelée **écart type quadratique moyen.** Dans notre cas, l'écart type étant de 3.157143 F/kg, sur une moyenne de 4.70 F/kg, nous pouvons affirmer que la variation du prix n'est pas négligeable (un peu plus de 15 %). Le programme listé ci-dessous calcule la moyenne et l'écart type d'un certain nombre de valeurs (MAX) saisies au clavier. Le programme mémorise les données dans le tableau A(100) ; (MAX ne peut être supérieur à 100).

PROGRAMME DE CALCUL DE L'ECART TYPE

```
1 REM * programme de calcul de la moyenne et de l'ecart type *
2 REM execute sur HHC en Microsoft BASIC
3 REM
10 DIM A(100)
20 INPUT "nombre d'observations";MAX
30 IF MAX=0 OR MAX>100 THEN PRINT "erreur":GOTO 20
40 T=0
50 FOR I=1 TO MAX
60 PRINT "observation no. ";I
70 INPUT "valeur ";A(I)
80 T=T+A(I):REM calcul de la somme
90 NEXT I
100 TMOY=T/MAX
110 PRINT #2"moyenne = ";TMOY
120 REM calcul de la somme des carres
130 T=0:REM T est reutilise avec une autre signification
140 FOR I=1 TO MAX
150 T=T+A(I)*A(I):REM T contient la somme des donnees au carre
160 NEXT I
170 PRINT #2"somme des carrés = ";T
180 S=SQR (T/MAX-(TMOY*TMOY)):REM S=ecart type
190 PRINT #2"ecart type      = ";S
200 REM
210 REM calcul du nombre de donnees dans l'intervalle
220 REM d'amplitude S et centre sur TMOY
230 INF=TMOY-S
240 SUP=TMOY+S
245 K=0
250 FOR I=1 TO MAX
260 IF A(I)>=INF AND A(I)<=SUP THEN K=K+1
270 NEXT I
280 REM
290 PRINT #2"on compte ";K;" valeurs dans"
300 PRINT #2"l'intervalle (TMOY-S;TMOY+S)"
310 NP=K*100/MAX
320 PRINT #2"soit un pourcentage de ";NP ;" %"
330 REM
340 INPUT "voulez-vous continuer (oui/non) ";REP$
350 IF REP$="oui" GOTO 20
360 END
```

```
Les donnees saisies sont celles de l'exemple precedent

moyenne = 4.7
somme des carres = 132.53
ecart type      = 3.157143
on compte 5 valeurs dans
l'intervalle (TMOY-S;TMOY+S)
```

Solutions du test 15

1 / a) La fonction doit être définie comme une chaîne, car elle utilise CHR$, qui renvoie une chaîne comme résultat. Il faut donc écrire :
DEF FNA$=CHR$(27)+CHR$(10) ;
d) Cette fonction est erronée, car elle contient une valeur numérique, alors qu'elle est définie en tant que chaîne ; la forme correcte est donc la suivante :
DEF FNA$="Chaîne d'essai"+"="+STR$(256).

---

2 / Nous proposons le programme suivant :

```
10  ' ** DEBUT
20  INPUT "CHAINE, NOMBRE ";A$,N
30  IF N<0 OR N>99 GOTO 20          ' CONTROLE
40  IF A$="FIN" GOTO 190
50  ' ** CONTROLER SI N PEUT ETRE UNE LETTRE EN ASCII
60  ' LETTRES MAJUSCULES SEULEMENT
70  K=0
80  IF N>65 AND N<90 THEN K=1
90  IF K=0 GOTO 20
100 '
110 '
120 B$=CHR$(N)
130 '
140 M=INSTR(A$, B$)
150 IF M=0 GOTO 20
160 PRINT "LE NOMBRE ";N;"REPRESENTANT LE CARACTERE ";B$
170 PRINT "SE TROUVE EN POSITION ";M;"DE LA CHAINE ";A$
180 GOTO 20
190 END
```

---

3 / Le programme doit utiliser des tableaux et des DATA. Voici une des formes possibles.

```
10  ' ** DEBUT
20  DIM CODES(11) ,NMT(10) ,NOM$(10)
30  FOR I=1 TO 10:READ NOM$(I) :NEXT I
40  DATA "PREMIER NOM"
50  '
60  '                    NOMS PREVUS
70  '
130 DATA "DIXIEME NOM"
140 ' LECTURE DONNEES
150 INPUT "CODE ";CD
155 IF CD=0 GOTO 310     ' IMPRESSION ET FIN DE PROGRAMME
160 IF CD<10 OR CD>20 GOTO 150
170 INPUT "MONTANT ";MT
180 IF MT<1250 OR MT>5700 GOTO 170
190 INPUT "QUANTITE ";QT
200 INPUT "NOM ";NM$
210 K=0
220 FOR I=1 TO 10
230 IF NM$=NOM$(I) THEN K=1
240 NEXT I
250 IF K=0 GOTO 200      ' K=0 SIGNIFIE QUE LE NOM EST INCORRECT
260 TOT=QT * MT          ' MONTANT TOTAL
270 L=CD-9
```

```
271  ' LE CODE (CD) A UNE VALEUR COMPRISE ENTRE 10 ET 20. LA DIFFERENCE
272  ' (CD-9) VARIE ENTRE 1 ET 11 (10-9=1 , 20-9=11) ET INDIQUE DONC
273    A QUELLE POSITION DU TABLEAU (CODES) CORRESPOND UNE VALEUR
274  ' DONNEE DE (CD) , PAR EXEMPLE, SI CD=12, L'INDICE (L) VAUT
275  ' 12-9=3 : LE MONTANT ASSOCIE AU CODE (CD) DOIT ETRE
276  ' MEMORISE EN POSITION 3 DU TABLEAU (CODES)
280  CODES (L) = CODES (L)+TOT      ' ACCUMULE LE TOTAL PAR CODES
290  NMT (K)=NMT (K)+TOT            ' ACCUMULE LE TOTAL PAR NOMS
291  ' CAR (K) POINTE SUR LE NOM (INSTRUCTIONS 210-240)
300  GOTO 150          ' SAISIE D'UN NOUVEAU GROUPE DE DONNEES
310  ' ** IMPRESSION **
320  FOR I=1 TO 11          ' SELECTIONNE UN DES 11 TOTAUX
330  N =I+9                 ' RESTITUE LE CODE
340  PRINT "CODE: ";N;TOTAL: ";CODES(I)
350  NEXT I
360  '
370  ' ** IMPRESSION PAR NOM **
380  FOR I=1 TO 10          ' SELECTIONNE UN DES 10 NOMS
390  PRINT "NOM: ";NOMS$(I) : "TOTAL: ",NMT(I)
400  NEXT I
410  END
```

---

4 / On obtient le coût unitaire moyen en additionnant tous les montants lus à la ligne 170, et en divisant ce total par le nombre de saisies effectuées. Le sous-programme doit prévoir un compteur qui s'incrémente à chaque saisie et un totaliseur qui cumulera tous les montants. A la fin du programme, leur quotient donne la valeur moyenne. Ce sous-programme sera appelé en insérant dans le programme précédent la ligne :

```
185  GOSUB 1000
```

et se résume à deux instructions :

```
1000  '*ENTREE
1010  COMPT= COMPT+1 'Compteur
1020  SOMME= SOMME+CT 'Totaliseur
```

A la fin du programme, on calcule la valeur moyenne recherchée et l'on affiche :

```
405  PRINT "Coût unitaire moyen = ";SOMME/COMPT
```

---

5 / L'algorithme consiste à exécuter une boucle de multiplication, de 1 à NUM, avec :
NUM = Valeur transmise au sous-programme, dont on veut calculer la factorielle.

```
1000  'Calcul de la factorielle du nombre NUM
1010  T=1 'Première valeur
1015  IF NUM=0 GOTO 1050 'Calcul de factorielle 0
1020  FOR I=2 TO NUM
1030  T=T*I
1040  NEXT I
1050  RETURN
```

En sortie, la variable T contient la factorielle de NUM.

---

6 / On utilisera ici le sous-programme précédent pour calculer les trois factorielles [N!, K! et (N-K)!] qui apparaissent dans la formule.

```
10  '** DEBUT
15  DEFDBL R-T
```

```
20  INPUT "Nombre d'éléments";N
30  INPUT "Nombre d'éléments par groupe";K
40  NUM=N
50  GOSUB 1000
60  S=T       'Sauvegarde dans S le résultat du sous-programme (factorielle de N)
70  NUM=K
80  GOSUB 1000
90  S1=T      'S1 contient la factorielle de K
100 NUM=N-K
110 GOSUB 1000
120 S2=T      'S2=(N-K)!
130 R=S/S1/S;      'R est le nombre de combinaisons cherché
140 PRINT "Il y a ";R;"manières de choisir"
145 PRINT K;"éléments parmi ";N
150 END
```

On remarquera que la ligne 130, qui exécute le calcul final, aurait pu s'écrire R=S(S1*S2). Cependant, à cause des ordres de grandeur des nombres S1 et S2, leur produit pourrait conduire à une valeur dépassant la précision de la machine. En effectuant d'abord la division S/S1, on réduit les risques d'erreur. De plus, même en déclarant les variables en double longueur (ligne 15), le nombre N que l'on peut traiter est limité à 30 (pour des valeurs supérieures, on obtient des erreurs de précision).

Les fonctions d'entrées/sorties des données

Dans ce chapitre sont exposées les commandes et les fonctions de saisie des données au clavier, d'affichage à l'écran ou d'édition sur imprimante. Les commandes et les fonctions d'E/S s'adressant aux unités disque ou aux bandes magnétiques seront traitées à part.

Dans tous les langages de haut niveau, les données à saisir ou à éditer doivent d'abord être traitées par des programmes système. En entrée, ceux-ci transposent la donnée générée par le clavier dans le format requis par la machine ; en sortie, ils adaptent le format des caractères aux particularités des périphériques. Les modes de conversion dépendent du type de donnée saisie : pour traiter un nombre entier, par exemple, les programmes système le convertissent en binaire et le stockent dans deux octets contigus en mémoire. Dans le cas de nombres réels, le processus est plus complexe. Dans la machine, les réels sont représentés sous forme exponentielle (floating point) et les programmes d'E/S assurent la distinction entre la mantisse et l'exposant. La position automatique du point décimal et le nombre de chiffres significatifs dépendent du type de machine et du système d'exploitation.

Le programme de saisie des données, en plus de sa fonction de « formatage », exécute tous les contrôles nécessaires afin d'assurer l'exactitude formelle des saisies : il vérifiera la cohérence entre la donnée fournie et le type demandé, et, dans le cas d'une saisie de plusieurs valeurs, entre le nombre de données saisies et le nombre attendu.

Les données à imprimer (ou à afficher à l'écran) sont extraites de la mémoire, modifiées éventuellement par le programme d'application, puis envoyées au périphérique. Cette adaptation formelle des données pour respecter une présentation particulière lors de l'édition s'appelle le **formatage** (du terme anglais format). Les formats d'acquisition ou d'impression des données sont définis par le programmeur, avec les instructions et les fonctions appropriées. En Basic, on peut également gérer les contrôles à effectuer, grâce à l'instruction ON ERROR GOTO ... étudiée précédemment.

Fonctions de saisie des données

Chaque phase de saisie s'achève en frappant la touche de validation CR (ou RETURN). Si les données saisies sont correctes, le système les transmet au programme d'application ; dans le cas contraire, un message approprié s'affiche à l'écran.

En Basic, on peut contrôler dans le programme certaines erreurs commises en phase de saisie ; dans tous les autres langages, celles-ci entraînent systématiquement l'arrêt de l'exécution.

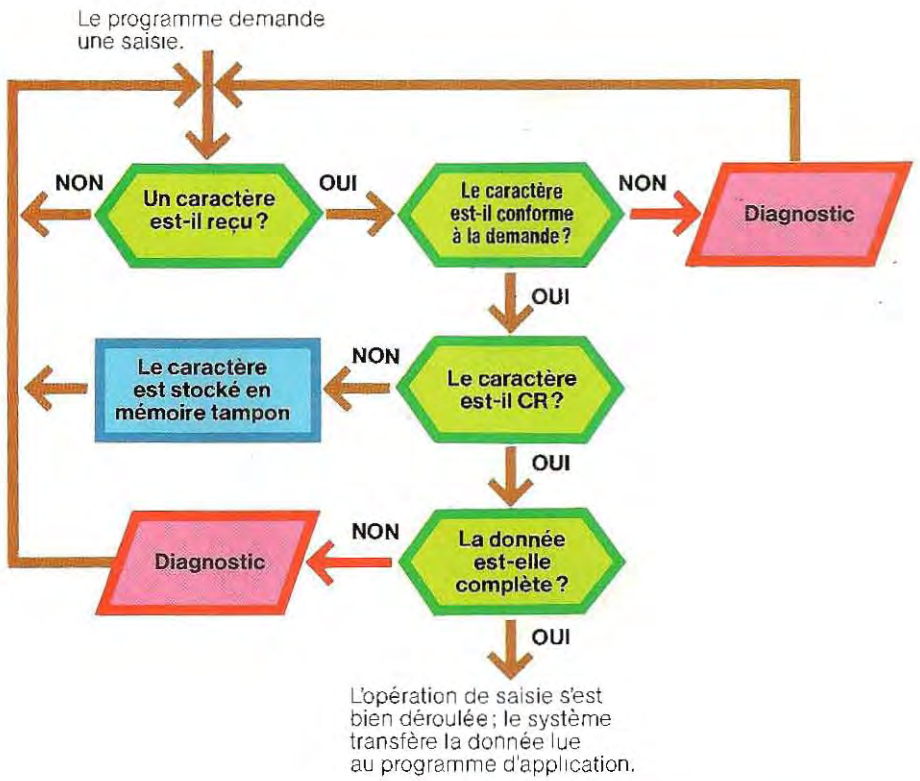
Le schéma ci-dessous illustre les fonctions exécutées en phase d'acquisition des données. Lorsqu'une instruction de lecture s'exécute, le contrôle est passé au programme système correspondant, qui se met en attente d'un caractère. Le temps machine nécessaire à la lecture d'un octet sur le port d'entrée est très court, comparativement au temps que met l'opérateur pour taper un caractère. Le système entre donc dans une boucle d'attente, comme le montre notre schéma. Une fois le caractère reçu, un premier contrôle formel est effectué. Par exemple, lorsqu'on demande une valeur numérique, le système vérifie qu'il ne s'agit pas d'une lettre. Le caractère est alors stocké dans un buffer (mémoire tampon), et le processeur reprend la boucle d'attente. Les caractères émis par l'opérateur sont rangés l'un derrière l'autre, de façon à former la donnée complète, jusqu'à ce que la saisie soit validée par la touche CR (ou RETURN). Le programme système vérifie alors que toutes les données demandées par l'instruction de lecture sont présentes ; dans l'affirmative, l'exécution du programme d'application reprend, sinon le système se met en attente des autres données nécessaires à l'achèvement de la requête.

INPUT. La principale instruction de lecture est la suivante :

INPUT "Commentaires"; VARIABLES

La chaîne "Commentaires", qui est facultative, représente un message que l'on désire affi-

SCHEMA DU PRINCIPE DE SAISIE DES DONNEES



cher au moment de la saisie. Le système écrit à l'écran la chaîne figurant entre guillemets, suivie d'un point d'interrogation, indiquant qu'il attend une saisie. L'opérateur peut alors entrer ses données, qui sont successivement affectées aux variables spécifiées dans l'instruction. Par exemple, l'exécution de :

```
INPUT "Essai"; V1,K$,B
```

provoquera l'affichage de la ligne :

```
Essai ?
```

L'opérateur devra entrer trois données (affectées respectivement à V1, K$ et B) : la première et la troisième seront des valeurs numériques, alors que la deuxième représentera une chaîne. Elles seront séparées par des virgules et, à la fin de la saisie, l'opérateur devra frapper la touche CR.
La saisie 3.4,ABC,100 satisfait l'instruction précédente. En revanche, si l'on intervertit l'ordre des données (par exemple 3.4,100,ABC), la cohérence avec le format demandé (nombre, chaîne, nombre) n'est plus respectée, et le programme système signale une erreur.
En phase de saisie, on peut supprimer le point d'interrogation émis par le système, en remplaçant le symbole ; par , . L'instruction

```
INPUT "Essai", V1,K$,B
```

produit le même effet que la précédente mais avec la suppression du symbole ? après le message. Normalement, le caractère CR (Carriage Return) signale à la machine la fin de la saisie et, en même temps, positionne le curseur au début de la ligne suivante. La dernière option acceptée par l'instruction INPUT permet d'empêcher le déplacement du curseur afin de disposer plusieurs entrées sur une même ligne. Pour cela, on insère le symbole ; tout de suite après le code :

```
INPUT ; "Description";V1,K$,B
```

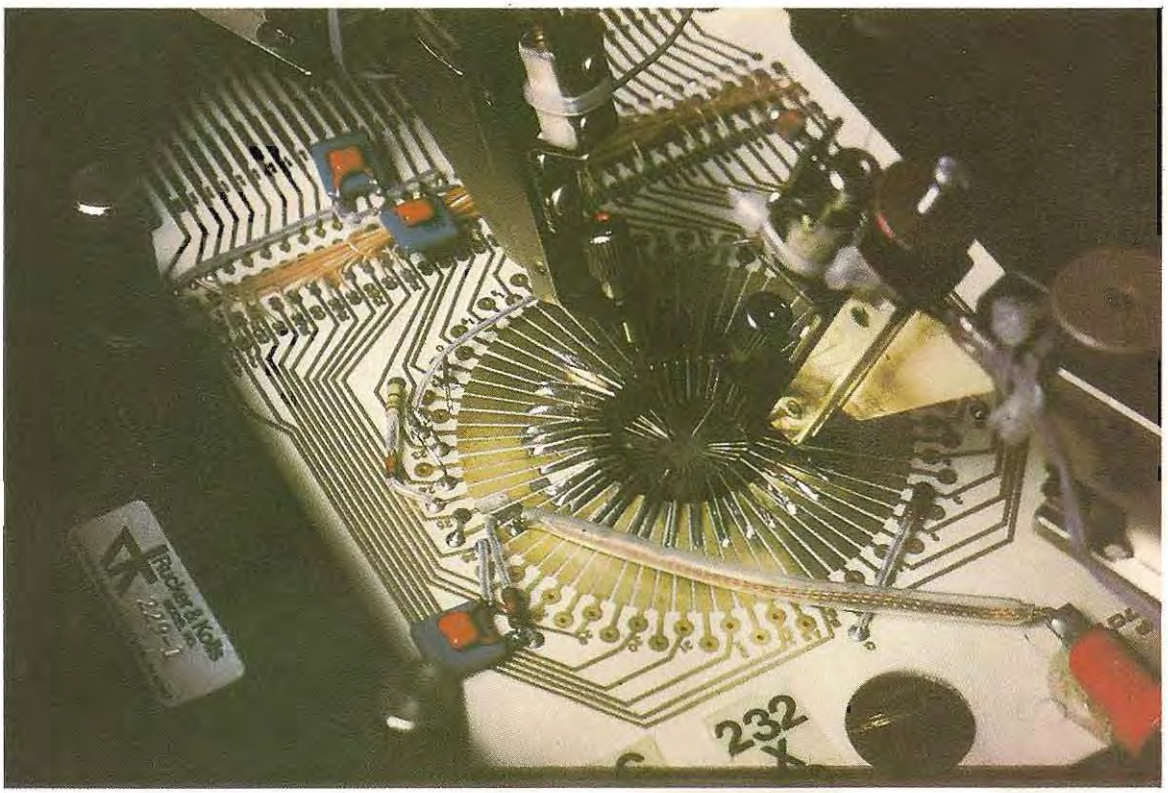
Cette formulation supprime le retour du curseur en début de ligne à la fin de la saisie.
L'instruction INPUT, sous l'une quelconque de ses formes, implique trois limitations :

1 / les caractères saisis sont écrits automatiquement sur l'écran (écho) ;

2 / la longueur de chaque donnée (nombre de chiffres pour les variables numériques ou de caractères pour les chaînes) n'est pas contrôlée par le système (sauf en cas de débordement) ;

3 / les caractères spéciaux (CR, RUBOUT, ESCAPE, etc.) ne peuvent pas être saisis.

Pendant la saisie des données, chaque caractère est acquis directement par le programme, tandis que son « écho » se répercute automatiquement à l'écran. Dans certains cas, par exemple pour saisir un mot de passe de fichier protégé, on préférera un autre type d'instruction de lecture, qui ne provoquera pas d'écho à l'écran.
Le deuxième point (contrôle de la longueur des données) s'avère crucial pour limiter les erreurs de saisie. C'est pourquoi la gestion par **fenêtres d'édition** est souvent utilisée. Avec ce mode de saisie, l'opérateur dispose d'un espace bien défini, dans lequel il peut taper ses valeurs ; ces dernières ne peuvent en aucun cas dépasser la longueur prévue. La simple instruction INPUT n'autorise aucune vérification de ce type. Au contraire, en contrôlant la longueur de la donnée au fur et à mesure de sa saisie, on peut interrompre la lecture au moment voulu.
Le dernier point mentionné est également lié à des utilisations particulières. En phase de lecture, l'instruction INPUT vérifie que les valeurs saisies et leurs variables associées sont cohérentes ; elle élimine d'éventuels caractères spéciaux, qui pourraient cependant s'avérer très utiles dans certaines applications, pour communiquer à l'ordinateur des ordres à la place de données. Prenons l'exemple simple d'un programme destiné à lire, puis à stocker dans un fichier, des renseignements relatifs au personnel d'une entreprise (nom, service, salaire, promotion, absences, primes). Or, nous sommes en présence d'un nouvel employé, et les trois dernières rubriques sont encore inconnues. Si aucun code ne peut imposer la fin de la saisie, l'utilisateur sera toujours contraint de compléter tous les champs demandés, et perdra peut-être un temps considérable.
Si, au contraire, on prévoit la possibilité d'interrompre la phase de saisie, on ne fournira à chaque fois que les données voulues (dans notre exemple, les trois premières). L'instruc-

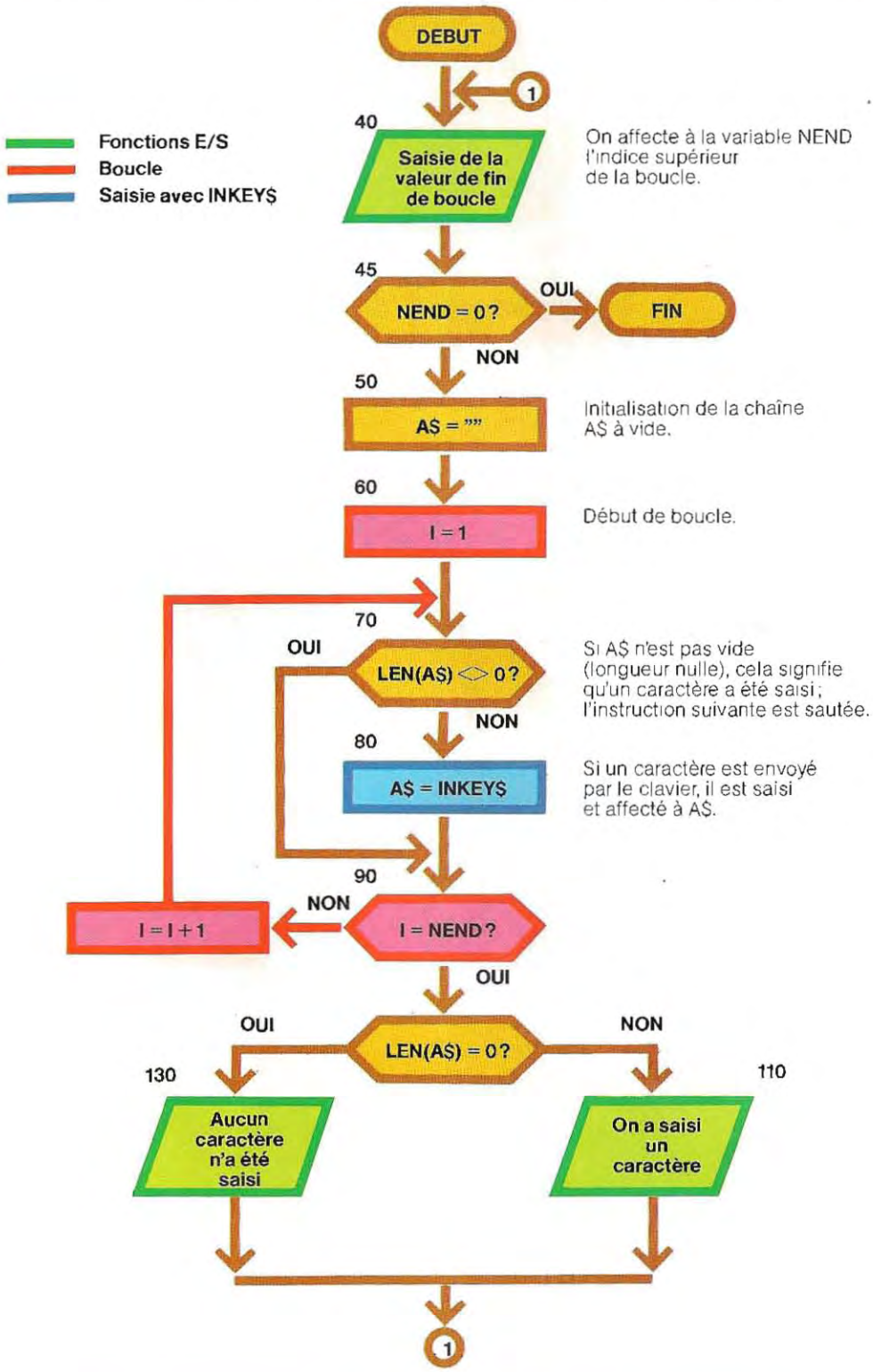


K. Reese/Marka

Contrôle d'un micro-circuit par ordinateur. Les barettes de cuivre rayonnantes établissent les connexions entre le chip (au centre) et l'extérieur.

tion INPUT précédente requiert dans tous les cas six valeurs (deux chaînes, un nombre, une chaîne, deux nombres) et on ne peut omettre aucune donnée.
L'opérateur doit entrer les différentes valeurs conformément à ce qui a été spécifié dans l'instruction de lecture.
Ces difficultés peuvent, dans une certaine mesure, être surmontées avec les instructions INKEY$ et INPUT$(N).
Cependant, de nouvelles complications apparaissent : tous les contrôles (vérifiant par exemple que la donnée est bien du type voulu), qui étaient auparavant effectués par les programmes système avec l'instruction INPUT, doivent dorénavant être assumés par le programme d'application.
Le choix d'une instruction de lecture particulière dépend de l'application envisagée. Dans un programme simple, qui saisit peu de données, l'instruction INPUT suffit. Au contraire, pour un programme d'une certaine complexité, nécessitant une bonne gestion de l'écran et des contrôles spécifiques, l'utilisation des autres formes devient indispensable.

EXEMPLE DE FONCTIONNEMENT DE L'INSTRUCTION INKEY$



EXEMPLE D'UTILISATION DE LA FONCTION INKEY$

```
10 '
20 ' *** EXEMPLE D'UTILISATION DE LA FONCTION INKEY$
30 '       FICHIER : SUNCAR
40 INPUT "Nombre de fois qu'il faut executer la boucle ";NEND
45 IF NEND=0 GOTO 150    ' Sortie
50 A$=""
60 FOR I=1 TO NEND
70 IF LEN(A$)<>0 GOTO 90
80 A$=INKEY$
90 NEXT I
100 IF LEN(A$)=0 GOTO 130
110 PRINT    "Saisie d'un caractere pendant la boucle"
120 GOTO 40
130 PRINT    "ON N'A SAISI AUCUN CARACTERE"
140 GOTO 40
150 END
```

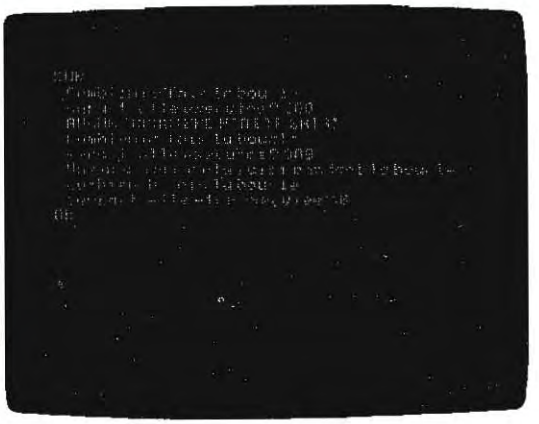
**INKEY$**. Permet la saisie d'un unique caractère depuis le clavier ; le caractère émis n'a pas d'écho et n'apparaît donc pas à l'écran. L'instruction INKEY$ accepte n'importe quel caractère (y compris les touches de contrôle), excepté CTRL + C (touche CONTROL et touche C enfoncées simultanément) qui arrête l'exécution du programme. Notons que certains Basics compilés acceptent même le caractère CTRL + C, qui est rendu temporairement transparent par l'instruction INKEY$. La syntaxe de l'instruction est :

A$ = INKEY$

Le caractère lu depuis le clavier est affecté à la chaîne A$, définie par l'utilisateur. Cette instruction de saisie ne met pas le système en attente d'une donnée : si un caractère est disponible, il est lu et affecté à la chaîne spécifiée dans l'instruction. Sinon, le programme poursuit son exécution. L'organigramme de la page 551 et le listing correspondant retracent un exemple de programme utilisant l'instruction INKEY$ dans une boucle d'attente. A chaque passage dans la boucle, INKEY$ examine l'état de l'unité d'entrée (clavier) et, si elle trouve un caractère prêt (pending), le transfère dans la chaîne A$. A partir de cet instant, la longueur de A$ n'est plus nulle (elle contient le caractère lu) : la condition figurant à l'instruction 70 [LEN(A$)<>0] est vérifiée, et l'instruction INKEY$ est dorénavant sautée. Au terme de la boucle, la chaîne A$ contiendra un caractère, ou bien sera encore vide (si l'opérateur n'a rien tapé).

UTILISATION DE LA FONCTION INKEY$

La fonction d'entrée INKEY$ permet d'acquérir un caractère depuis le périphérique (écran). Cette page retrace l'exécution du programme montré en exemple à la page précédente.



■ Avec la commande RUN, l'exécution démarre en ligne 40 (instruction de lecture).

```
RUN
Nombre de fois qu'il faut
executer la boucle? 300
ON N'A SAISI AUCUN CARACTERE
Nombre de fois qu'il faut
executer la boucle? 300
Saisie d'un caractere pendant la boucle
Nombre de fois qu'il faut
executer la boucle? 0
Ok
```

L'exécution de cette ligne provoque l'affichage du message que le programmeur a inclus entre guillemets, suivi d'un point d'interrogation (ajouté par le système). L'opérateur tape le nombre 300 qui sera affecté par l'instruction INPUT à la variable NEND (limite supérieure de l'indice de boucle).

Dès que l'opérateur frappe la touche RETURN, après la saisie du nombre 300, l'exécution se poursuit en ligne 50 (initialisation de la chaîne A$ à vide) et se poursuit avec l'entrée dans la boucle (ligne 80) qui contient la fonction INKEY$.

■ Pendant le déroulement de la boucle, aucun caractère en attente n'a été trouvé sur le port d'entrée : la condition de la ligne 100 est vérifiée et on exécute l'instruction d'édition (ligne 130).

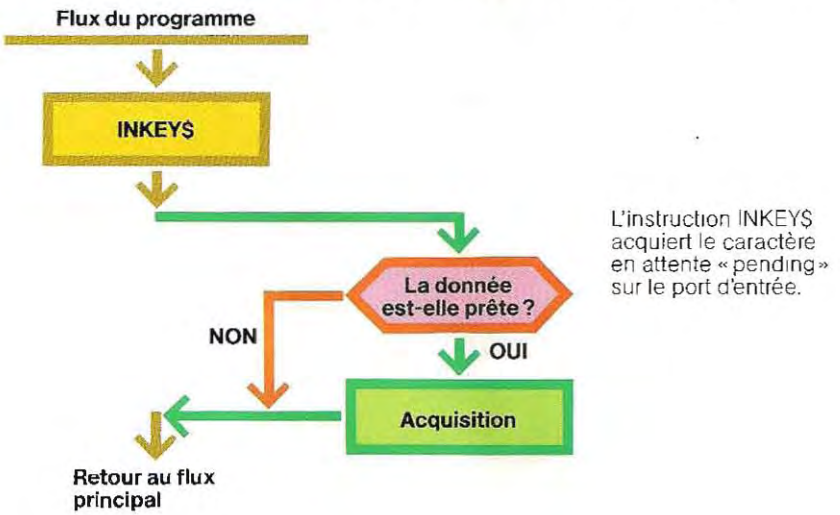
■ L'étape précédemment décrite se répète, mais cette fois l'opérateur a entré un caractère quelconque durant l'exécution de la boucle. Ce caractère n'apparaît pas à l'écran, car la fonction INKEY$ ne provoque pas d'écho, mais est affecté à A$. La condition de la ligne 100 est maintenant fausse et l'exécution passe à la ligne 110, qui provoque l'affichage du message.

■ Pour sortir de l'exécution, l'opérateur entre la valeur 0. Le test prévu en ligne 45 provoque un saut à la fin du programme.

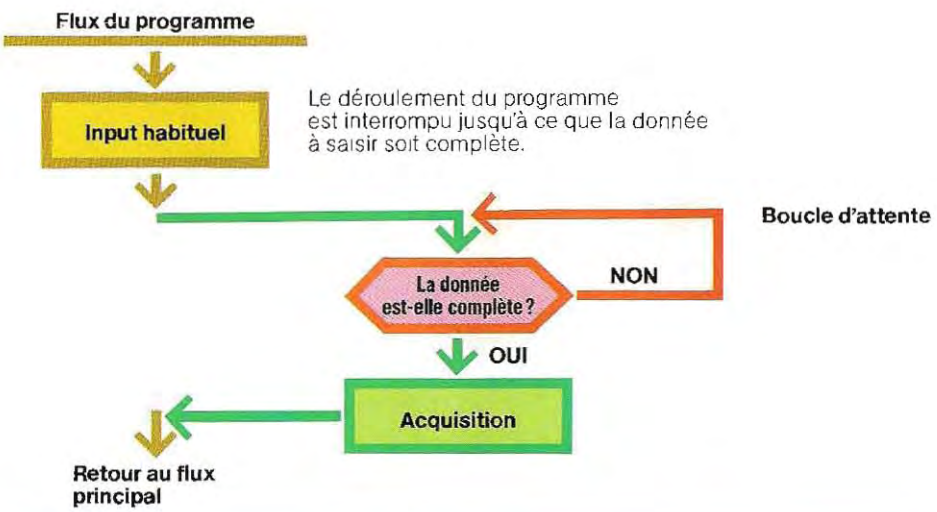
L'instruction INKEY$ fonctionne d'une façon radicalement différente des autres ordres de lecture. Si l'on remplaçait la ligne 80 (A$=INKEY$) par une des autres instructions d'entrée, l'exécution s'arrêterait à cette instruction, dans l'attente de la donnée. Avec l'instruction INKEY$, le système ne se met pas en attente : il poursuit son activité en cours, tout en gardant sous contrôle le port d'entrée, afin d'acquérir la donnée dès son introduction. Les schémas ci-dessous explicitent la logique des deux modes d'acquisition mis en œuvre.

INPUT$(N). Saisit une suite de N caractères depuis le clavier sans produire d'écho. Ainsi, A$=INPUT$(6) transfère six caractères dans la chaîne A$. Dans ce cas, il est inutile de frapper la touche CR pour mettre fin à la saisie : le système comptabilise, au fur et à mesure le nombre de caractères lus ; dès que la donnée atteint la longueur requise, l'exécution passe à l'instruction suivante.

FONCTIONNEMENT DE L'INSTRUCTION INKEY$



FONCTIONNEMENT DES AUTRES INSTRUCTIONS DE SAISIE



EXEMPLES D'UTILISATION DE LA FONCTION INPUT$(N)

```
10 ' ** EXEMPLES D'UTILISATION DE LA FONCTION INPUT$(N)
20 '    FICHIER : INPUT
30 DEFINT A-Z
40 INPUT "Nombre de caracteres a saisir ";N
50 IF N<=0 OR N>80 GOTO 40        ' CONTROLE DU NOMBRE N DE CARACTERES
60 '
70 INPUT "Echo demande (repondre OUI ou NON) ",REP$
80 '
90 ' Le format utilise ligne 70 (symbole "," et non ".") supprime
100 ' le point d'interrogation affiche par le systeme
110 '
120 IF REP$<>"OUI" AND REP$<>"NON" GOTO 70     ' CONTROLE
130 PRINT "Entrez ";N;" caracteres"
140 A$=INPUT$(N)
150 IF REP$="OUI" THEN PRINT A$
160 '
170 ' * SECONDE METHODE
180 '
190 PRINT "Seconde methode : A$ est reecrit caractere par caractere"
200 A$=""
210 FOR I=1 TO N
220 B$=INPUT$(1)     ' LECTURE D'UN SEUL CARACTERE A LA FOIS
230 PRINT B$;
240 A$=A$+B$     ' LE CARACTERE SAISI EST CONCATENE A A$
250 NEXT I
260 '
270 INPUT "VOULEZ-VOUS CONTINUER (OUI/NON)";REP$
280 IF REP$="OUI" GOTO 40
290 END
```

La fonction INPUT$(N) (il ne s'agit pas d'une instruction) ne produit pas d'écho ; il faut donc, pour visualiser les caractères entrés, le spécifier au moyen d'une instruction insérée dans le programme.
L'exemple listé ci-dessus utilise la fonction INPUT$(N), et prévoit les deux formes possibles (avec ou sans écho). Pour obtenir un écho, il faut lire puis envoyer à l'écran un seul caractère à la fois, de façon à ce que la saisie soit immédiatement suivie de son écho (seconde méthode).

LINE INPUT. Permet d'acquérir une ligne entière de caractères (normalement jusqu'à un maximum de 256). Par exemple, l'instruction

LINE INPUT "Commentaire";A$

transfère dans A$ une chaîne saisie au clavier.

Fonctions de sortie des données

Pour l'impression ou l'affichage à l'écran, le système réalise un transcodage des données (par exemple de binaire en ASCII), et les formate selon les instructions de l'utilisateur.
Il y a plusieurs façons d'écrire des données ; les valeurs numériques peuvent comporter un nombre variable de décimales, ou nécessitent parfois des caractères spéciaux. Chacun des aspects sous lesquels une même donnée peut se présenter à l'impression constitue un **format**. L'utilisateur dispose d'instructions appropriées pour spécifier le format qu'il souhaite ; en cas de sortie non formatée (aucune instruction explicite), le système adopte par défaut la représentation exponentielle pour les données numériques, et écrit les chaînes sans les modifier.
Rappelons que le format exponentiel d'une valeur numérique se compose d'une mantisse (chiffres significatifs du nombre en question, séparés par un éventuel point décimal) suivie du symbole E±nn, où E indique qu'il faut multiplier la mantisse par une puissance de 10, nn représentant l'exposant de celle-ci. Si l'exposant est positif, le signe + peut être omis. Ainsi, la notation 4931.3579E2 représente le nombre 493135.79 (4931.3579 × 100, car E2 vaut 100).
Les instructions les plus simples d'émission de données sont PRINT (pour l'affichage à l'écran) et LPRINT (pour la sortie sur imprimante). Pour utiliser un format particulier, l'instruction devient PRINT USING « spécification de format » (« imprime en utilisant... »)

pour l'écran, et LPRINT USING... pour l'imprimante.
Remarquons cependant que l'écran possède seulement 80 colonnes (si nous excluons les consoles graphiques), alors que l'imprimante tolère jusqu'à 132 caractères, et parfois plus. Des problèmes de longueur de ligne peuvent donc se présenter : ils se résoudront en répartissant sur plusieurs lignes de l'écran les données à éditer.

LPRINT. La formulation habituelle de cette instruction est :

LPRINT "Commentaires" ; VARIABLE1 ; ...

Le Basic imprime le message spécifié entre guillemets (symboles "), puis les valeurs des variables énumérées dans l'instruction (VARIABLE1, etc.) Ainsi l'exécution de :

```
10  V1=3.516
20  V2=75
30  LPRINT "Sortie=",V1,V2
```

provoque l'impression de la ligne :

```
Sortie=3.516      75
```

Les valeurs numériques ne sont écrites en format exponentiel (Enn) que si elles dépas-

EXEMPLES D'UTILISATION DE L'INSTRUCTION "LPRINT"

```
10 ' *** EXEMPLES D'UTILISATION DE L'INSTRUCTION (LPRINT) ***
20 '              FICHIER = LPR
30 A1=123
40 A2=456
50 A3=789
60 LPRINT "ESSAI N.1",A1,A2,A3
65 LPRINT                          ' ESPACEMENT
70 LPRINT "ESSAI N.2",A1,A2,A3
75 LPRINT                          ' ESPACEMENT
80 LPRINT "ESSAI N.3",A1,A2,A3
85 LPRINT                          ' ESPACEMENT
90 END
```

```
ESSAI N.1      123           456           789

ESSAI N.2      123  456  789

ESSAI N.3 123  456  789
```

```
10 ' *** EMPLOI DE LPRINT AVEC DES CHAINES **
20 '              FICHIER = LPRC
30 A$="PARIS"
40 B$="EST"
50 C$="GRAND"
60 LPRINT "ESSAI N.1";A$;B$;C$
70 LPRINT
80 LPRINT "*** ATTENTION : L'ESPACEMENT CORRECT EST
90 LPRINT "    PREVU PAR L'INSTRUCTION D'ECRITURE ***"
100 LPRINT "ESSAI N.2",A$,B$,C$
120 END
```

```
ESSAI N.1 PARIS EST GRAND

*** ATTENTION : L'ESPACEMENT CORRECT EST
    PREVU PAR L'INSTRUCTION D'ECRITURE ***

ESSAI N.2      PARIS          EST            GRAND
```

sent le nombre de chiffres maximum autorisé (suivant leur type d'appartenance, 6 chiffres pour les réels, 15 en double précision) ; dans notre exemple, l'exposant est donc omis.
A l'impression, chaque ligne est divisée en champs de 14 caractères. En séparant les variables par une virgule, chacune est imprimée au début de son champ correspondant ; au contraire, en utilisant le point-virgule comme symbole de séparation, toutes les variables seront imprimées à la suite. Si les données sont de type numérique, elles seront cependant espacées par un seul blanc.
Ces deux notations (, ou ;) peuvent également figurer à la fin de l'instruction ; en ce cas, l'exécution du LPRINT suivant continuera l'impression sur la ligne en cours. Tout se passe alors comme si la nouvelle instruction LPRINT était une continuation de la ligne précédente. La page 556 illustre quelques exemples d'édition de variables numériques et de chaînes.

TAB(N). Les fonctions d'émission de données, et surtout de sortie sur l'imprimante, nécessitent souvent une très grande liberté dans le positionnement des champs à éditer. Les instructions LPRINT (ou PRINT), avec leurs options ; et , offrent la possibilité d'éditer les données de manière contigüe ou de les répartir en champs de 14 caractères. Pour obtenir des tabulations plus précises, on emploie la fonction TAB(N), qui déplace de N caractères la zone d'édition de la donnée. Ainsi, l'instruction LPRINT TAB(20);B$ provoque l'impression de la chaîne contenue dans la variable B$ à partir de la position 20. Quelques exemples d'emploi de la fonction TAB(N) seront présentées par la suite, dans le chapitre sur la mise en page des données.

LPRINT USING. Permet l'impression de données numériques ou de chaînes, selon un format défini par l'utilisateur. Voici la syntaxe :

LPRINT USING"Spécification de format";VARIABLES

Les spécifications de format établissent, au moyen de symboles conventionnels, des modes particuliers d'édition.
Les formats prévus par le Basic 80 sont réunis ci-dessous.

■ Variable de chaîne

```
"!"
```

n'imprime que le premier caractère de la chaîne.

```
"\ \"
```

imprime un nombre de caractères égal au nombre d'espaces compris entre les deux symboles \ plus deux.
Par exemple, les lignes

```
10 A$="ESSAI D'EDITION"
20 LPRINT USING "\ \";A$
30 ' 2 blancs entre les deux back slash
```

imprimeront 4 caractères (2 espaces + 2) de la chaîne A$.

```
"&"
```

imprime la chaîne sans contrôle de longueur.

■ Variables numériques

```
"##.#"
```

spécifie* le nombre de chiffres à imprimer, en fixant la longueur de la partie entière et de la partie décimale.
Le nombre de chiffres de la partie entière est égal au nombre des symboles # (dièse) qui précèdent le point décimal.
Par exemple, l'exécution de

```
10 R=35.716
20 LPRINT USING "##.#";R
```

provoque l'impression du nombre 35.7 (2 chiffres pour la partie entière, puis une décimale).

```
"+"
```

permet l'écriture du signe de la donnée avant ou après sa valeur numérique. Dans certaines applications commerciales, il est d'usage de placer le signe après le nombre ; ainsi, la valeur −519 s'imprimera 519−.
Par exemple, les lignes

```
10 R=125.365
20 PRINT USING "+####.##";R
30 PRINT USING "####.##+";R
```

* Sur certaines imprimantes, le symbole # apparaît sous la forme £.

fournissent deux sorties :
– 125.36 (ligne 20 : seules 2 décimales figurent) 125.36 – (ligne 30).

"—"

Placé obligatoirement à la fin de la chaîne décrivant le format ("##.#— "), il provoque l'impression d'un éventuel signe — à la suite du nombre. Ce symbole est comparable à l'option "+" placée à la fin du champ, mais ici le signe n'est imprimé que s'il est négatif.

"**"

Le double astérisque remplace les éventuels espaces blancs par le symbole *. La partie entière du nombre à éditer peut quelquefois contenir un nombre de chiffres inférieur à celui prévu par le format "#". Si l'on ne spécifie pas d'autre option, il apparaîtra dans le champ d'édition autant d'espaces blancs que de chiffres manquants.
Par exemple, l'impression du nombre 7.1543 avec le format "####.##" donne en sortie 7.15 alors que par le format "**#####.##", elle donne ***7.15.

"$$"

imprime le symbole $ (dollar) juste avant le nombre. Avec ce format, l'éventuel signe négatif doit toujours figurer à la suite de la valeur numérique. Ainsi la spécification "$$—##.#" est erronée (signe — précédant le nombre); la forme correcte est "$$##.#—"

"**$"

Ce format est une combinaison des deux précédents. Il remplit les espaces blancs par des astérisques et place le symbole ($) de devise au début du nombre.

","

La virgule , utilisée pour mettre en évidence les milliers, selon la notation anglo-saxonne, est placée tous les trois chiffres de la partie entière, en partant de la droite. Dans les formats, elle doit toujours apparaître à gauche du point décimal. Ainsi la spécification "####,.##", appliquée au nombre 7563.121, produit en sortie : 7,563.12
L'unique variante autorisée consiste à placer la virgule à la fin du format lui-même (####.##,). Dans ce cas, le symbole , sera imprimé à la fin du nombre. L'impression du nombre précédent sera 7563.12,

"XXXX"

Quatre caractères quelconques, inclus dans un format, servent à spécifier la notation exponentielle : ils permettent de réserver l'espace nécessaire à la notation exponentielle E+nn (où nn représente l'exposant), qui est précisément constituée de 4 caractères.
Le format "##.#AAAA", appliqué au nombre 1200, produit l'impression de 12.0E+02 (E+02=10^2= 100;12.0×100=1200), puisque l'on demande seulement trois chiffres, dont une décimale (##.#), en format exponentiel (AAAA).

"%"

Ce symbole, lorsqu'il fait partie d'un format, apparaît devant la donnée, si elle dépasse la longueur du champ déclaré. Par exemple, le nombre 1470.65, imprimé avec le format "####.#", apparaît sous la forme %1470.6 car sa zone décimale déborde du champ prévu par le format.

Quelques exemples d'utilisation des différents formats sont programmés page 559. Remarquons que le système, en phase d'édition, arrondit automatiquement les données numériques dont la partie décimale excède en longueur celle du champ déclaré. Par exemple, le nombre 412.67, édité avec le format "###.#", devient 412.7 puisqu'on ne demande qu'une seule décimale.
L'arrondi se calcule selon les règles habituelles : si le chiffre à partir duquel s'effectue la troncature est supérieur à 5, on ajoute 1 au chiffre précédent ; sinon il est supprimé. Par exemple, avec un format autorisant une seule décimale, le nombre 56.86 sera transformé en 56.9 (le chiffre retiré étant 6), alors que 56.85 apparaîtra sous la forme de 56.8. Ainsi, une différence de 1/100 seulement entre les valeurs réelles (en mémoire) de deux variables peut devenir en sortie égale à 1/10, du fait des arrondis.

EXEMPLES D'UTILISATION DE L'INSTRUCTION "LPRINT USING"

```
10 ' ** EXEMPLES D'UTILISATION DE L'INSTRUCTION "LPRINT USING"
20 '
30 '              FICHIER = LPRUSI
40 ' *** OPTION "!" (IMPRIME SEULEMENT LE PREMIER CARACTERE DE LA CHAINE)
50 A$="PARIS ET LES PARISIENS"
60 LPRINT USING "!";A$
70 LPRINT                    ' ESPACEMENT
80 '
90 ' *** OPTION "£££.£" (VARIABLES NUMERIQUES)
100 R=123.789        ' N.B. LE NOMBRE POSSEDE UNE PARTIE ENTIERE A 3 CHIFFRES,
105 '                      ET TROIS DECIMALES
110 LPRINT USING "£££.£";R    ' IMPRIME (R) AVEC TROIS CHIFFRES
120 '                          POUR LA PARTIE ENTIERE ET UNE SEULE DECIMALE
130 LPRINT
140 '
150 ' *** OPTION "%"
160 LPRINT USING "%££.££";R    ' LE SYMBOLE % METTRA EN EVIDENCE QUE
170 '                           LE NOMBRE IMPRIME DEBORDE DU FORMAT D'IMPRESSION
180 LPRINT USING "%£££.££";R
190 LPRINT : LPRINT
200 '
210 ' *** OPTION "+£££.££"
220 LPRINT USING "+£££.££";R    ' IMPRIME (R) AVEC TROIS CHIFFRES POUR LA PARTIE
230 '                           ENTIERE, DEUX DECIMALES ET LE SIGNE A GAUCHE
240 ' *** OPTION "£££.££+"
250 LPRINT USING "£££.££+";R    ' COMME CI-DESSUS, MAIS AVEC LE SIGNE A DROITE
260 LPRINT
270 LPRINT USING "+£££.£££";R   ' IMPRIME (R) DANS UN FORMAT TEL QU'AUCUN
280 '                           CHIFFRE N'EST TRONQUE
290 LPRINT USING "£££.£££+";R   ' COMME CI-DESSUS, MAIS AVEC LE SIGNE A DROITE
300 LPRINT
310 '
320 ' OPTION "£££.£££-"         ' N.B. CE FORMAT FAIT APPARAITRE LE SIGNE
330 '                           A DROITE, AVEC OMISSION S'IL EST POSITIF
340 LPRINT Q=-456.789           ' (VALEUR NEGATIVE)
350 LPRINT USING "£££.£££-";Q
360 LPRINT USING "£££.£££-";Q
370 END
```

```
 R

 123.8

 %%123.79
 %123.79

 +123.79
 123.79+

 +123.789
 123.789+

 123.789
 456.789-
```

Si ces formats sont prévus dans la bibliothèque Basic Standard, tous les interpréteurs (ou compilateurs) ne les proposent cependant pas. En Basic Applesoft, par exemple, aucun formatage des données n'est possible : seule existe l'instruction TAB(N).

Les autres langages évolués, comme le Fortran ou le Pascal, offrent de nombreuses possibilités de formatage, à travers un jeu d'instructions très développé.

Test 16

1 / Une **matrice** est une structure mathématique qu'on peut représenter par un tableau à deux dimensions (lignes et colonnes). Par exemple, une matrice 4 × 5 (4 lignes, 5 colonnes) sera mémorisée sous la forme d'un tableau A (4,5).
Ecrire un programme qui saisit au clavier les cinq éléments de la première ligne de la matrice A [A(1,1), A(1,2)..., A(1,5)] et qui calcule les autres éléments, comme le produit du numéro de ligne à laquelle ils appartiennent par l'élément correspondant de la ligne 1. Par exemple, pour la ligne 2, on aura A(2,1)=2*(1,1), A(2,2)=2*A(1,2); en ligne 3, A(3,4)=3*A(1,4).

2 / Ecrire un sous-programme d'impression de la matrice A(4,5) calculée précédemment, soit de façon normale (ligne par ligne), soit de façon inverse (colonne par colonne).

3 / A l'aide de la fonction INPUT$(N), écrire un sous-programme de lecture (avec écho) d'un sigle formé de trois caractères, qui vérifie l'absence de caractère numérique.

4 / Généraliser la routine précédente en l'adaptant à la lecture de N caractères, compris entre 1 et 40.

Les solutions du test se trouvent page 567.

Structuration des impressions

Les sous-programmes d'impression peuvent être conçus de deux manières : en écrivant les instructions selon un format déterminé, ou en paramétrant la ligne à imprimer. La plus courante est la première méthode, qui consiste à écrire pour chaque application une routine spécifique contenant les instructions nécessaires, et qui, en aucun cas, ne s'adaptera à d'autres programmes. La seconde méthode, beaucoup plus complexe, consiste à préparer une variable de chaîne contenant toutes les informations que l'on désire obtenir sur une ligne d'impression. La routine d'impression se limite alors à l'émission de cette variable, appelée **vecteur d'impression**.
La première technique est rigide : pour changer un seul paramètre, il faut modifier le programme. La seconde peut devenir très exhaustive, au point d'obtenir un programme qui demande à l'utilisateur quelles sont les variables à imprimer et leurs positions respectives sur la page. Une telle généralisation demande une expérience d'analyse et de programmation considérable que nous illustrerons par quelques exemples très simplifiés.

Impressions non paramétrées. Voici un premier exemple. On désire imprimer un registre de clients, contenant les coordonnées décrites dans le tableau 1 et les données comptables du tableau 2 (voir page 561). Il faut prévoir un saut de page à chaque nouveau client (changement de code client) ou toutes les 30 lignes de données comptables, même si le client ne change pas. Les pages seront numérotées et, à la fin de l'édition relative à chaque client, on totalisera les montants nets (somme des champs Z2).
La première étape consiste à préparer le schéma d'impression. Sur une feuille quadrillée (il existe des feuilles prévues à cet effet), on reporte à la main les colonnes d'impression, les lignes concernant les coordonnées et les 30 lignes (maximum prévu pour chaque page) de données comptables, en les cadrant aux positions voulues, conformément à la mise en page que l'on désire obtenir.
De cette manière, la position de chaque champ d'impression est déterminée par les coordonnées (ligne et colonne) de la feuille quadrillée. La programmation consistera à traduire en instructions ce qui est représenté sur la feuille.
La page 562 illustre un exemple de disposition des champs sur 80 colonnes, destiné aux

imprimantes les plus petites ; on adaptera aisément cette grille à 132 colonnes, par exemple en espaçant davantage les champs. En haut de la page 563 est schématisé le programme principal, et en bas l'organigramme du sous-programme d'impression. Les fonctions exécutées par ce dernier sont :

1 / reconnaissance du code client ; au changement de code, on doit imprimer le total (Z3) relatif au client précédent, réinitialiser à zéro la variable Z3 pour y cumuler les nouvelles données, et imprimer les coordonnées du nouveau client ;

2 / reconnaissance de la première page ; le schéma précédent ne peut s'appliquer au premier client : en effet, aucun cumul de données précédentes (Z3) ne doit être imprimé. Le programme teste simplement le numéro de la page : si celui-ci est nul, l'instruction d'impression du total précédent est sautée, et l'en-tête seul est imprimé ;

3 / contrôle des lignes ; pour chaque ligne de donnée comptable imprimée, on incrémente un compteur. Dès que la valeur 31 est atteinte, on passe à une nouvelle page, même s'il s'agit toujours du même client.

Sur l'organigramme de la page 563 (en bas), on vérifie en effet que l'impression de l'en-tête et des coordonnées est nécessaire en deux points différents du programme. Il sera donc préférable d'utiliser un sous-programme au lieu de répéter les mêmes instructions. Cette structuration en routines distinctes présente un autre avantage : les différentes fonctions sont nettement séparées, facilitant la compréhension par le lecteur, et, de plus, les éventuels changements pour adapter le programme à de nouvelles applications en sont largement simplifiés.
La page 564 illustre l'organigramme de la routine d'impression de l'en-tête et des coordonnées. Ce sous-programme est activé par la rupture de séquence (changement de code client) ou après l'impression de 30 données comptables. Le programme est listé page 566. Dans cet exemple, la saisie des données s'effectue au clavier, mais dans les applications réelles, les données seront lues à partir de fichiers disque.

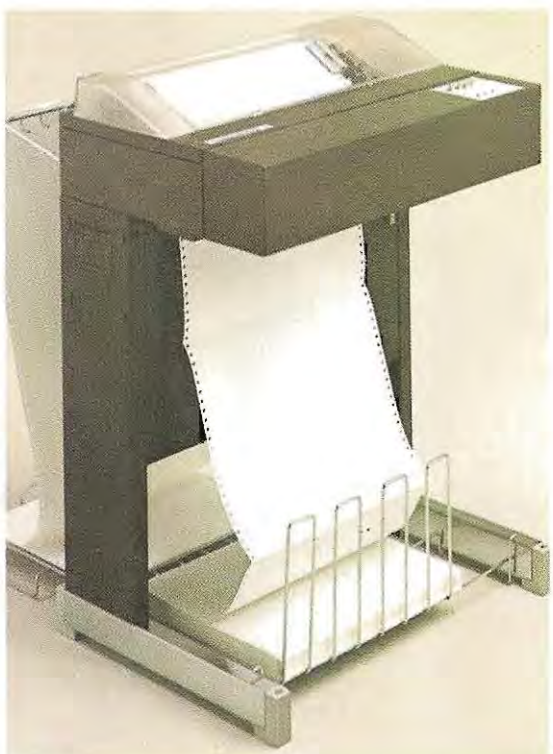
Tableau 1

Nom du champ	Signification	Type	Longueur
CC	Code client	numérique	3
NO$	Nom	alphabétique	15
RU$	Rue	alphabétique	10
NU	Numéro	numérique	5
TL	Téléphone	numérique	12 (indicatif et numéro)
CP	Code postal	numérique	5
VI$	Ville	alphabétique	15
DP$	Département	alphabétique	10

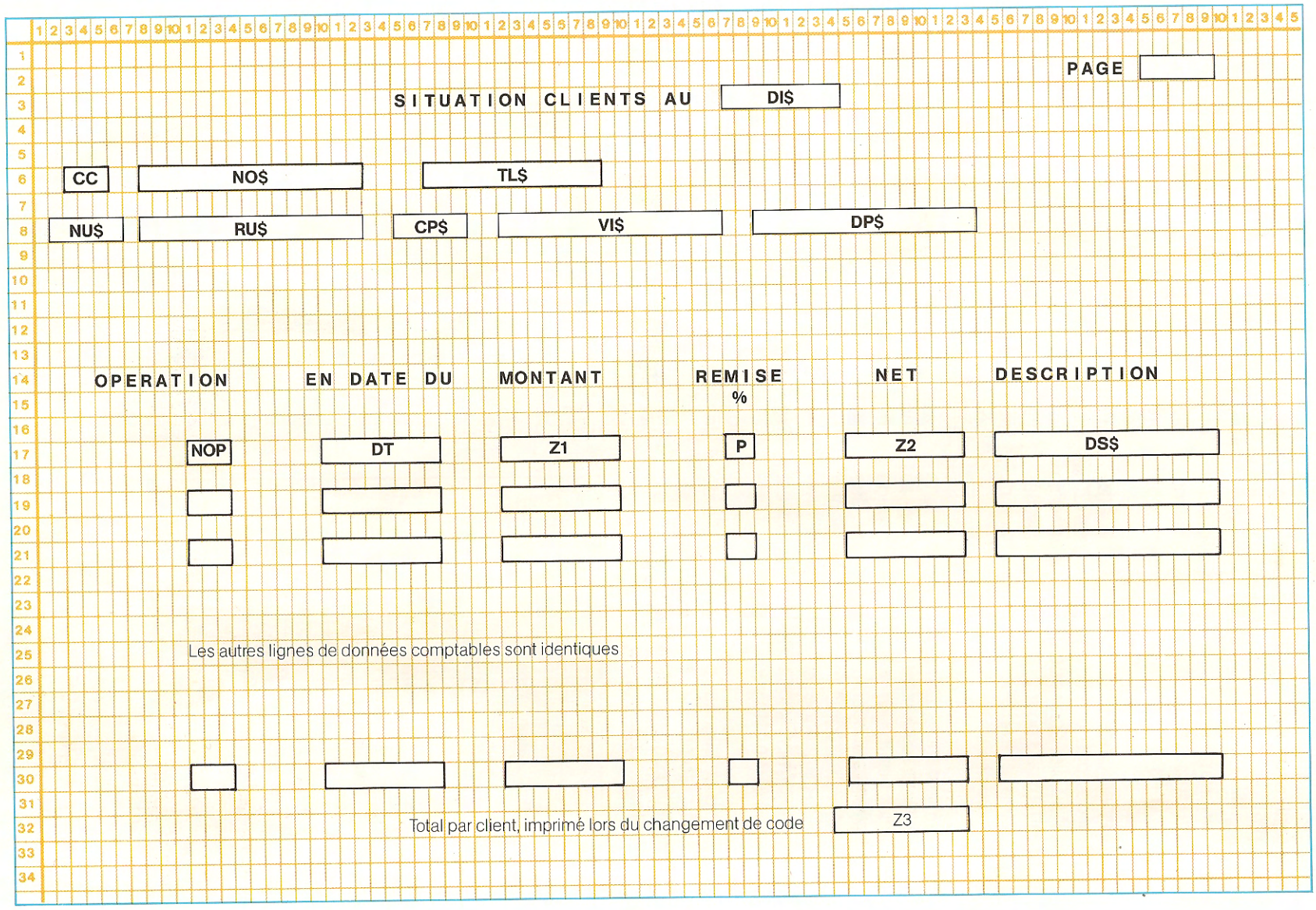
Tableau 2

Nom du champ	Signification	Type	Longueur
NOP	Numéro d'opération	numérique	3
DT	Date	numérique	8
Z1	Montant	numérique	8
P	Remise (%)	numérique	2
Z2	Montant net	numérique	8
DS$	Description	alphabétique	15

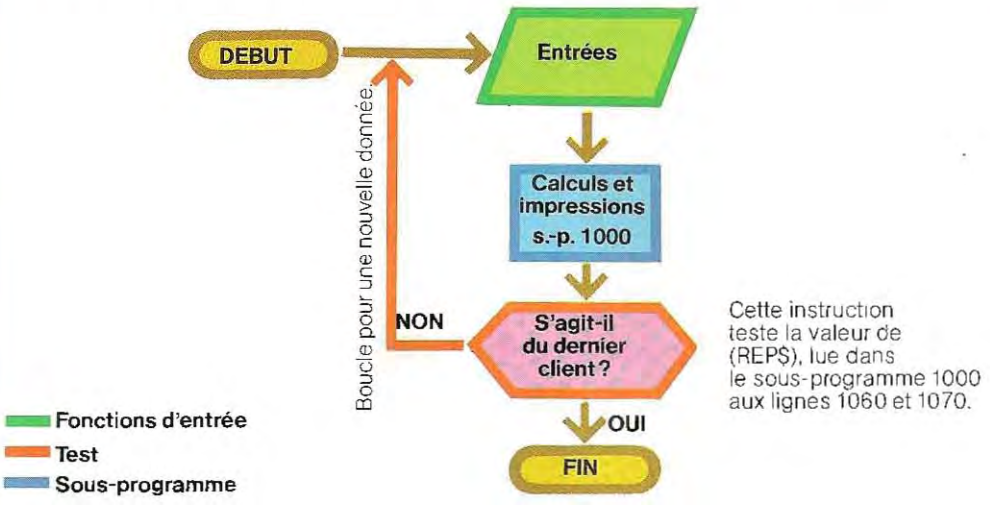
Imprimante bi-directionnelle. L'impression est sous le contrôle d'un microprocesseur.



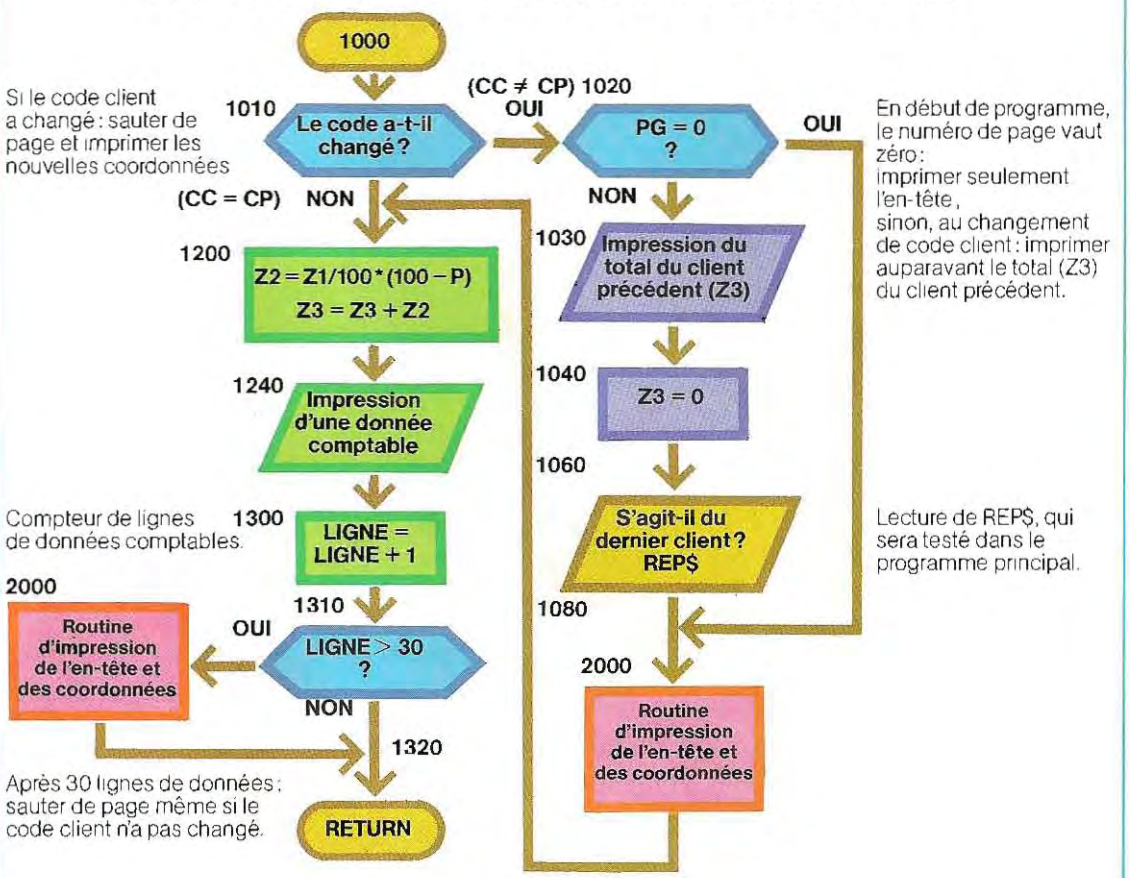
Olivetti


PAGE
SITUATION CLIENTS AU
DI$
CC
NO$
TL$
NU$
RU$
CP$
VI$
DP$
OPERATION
EN DATE DU
MONTANT
REMISE %
NET
DESCRIPTION
NOP
DT
Z1
P
Z2
DS$
Les autres lignes de données comptables sont identiques
Total par client, imprimé lors du changement de code
Z3

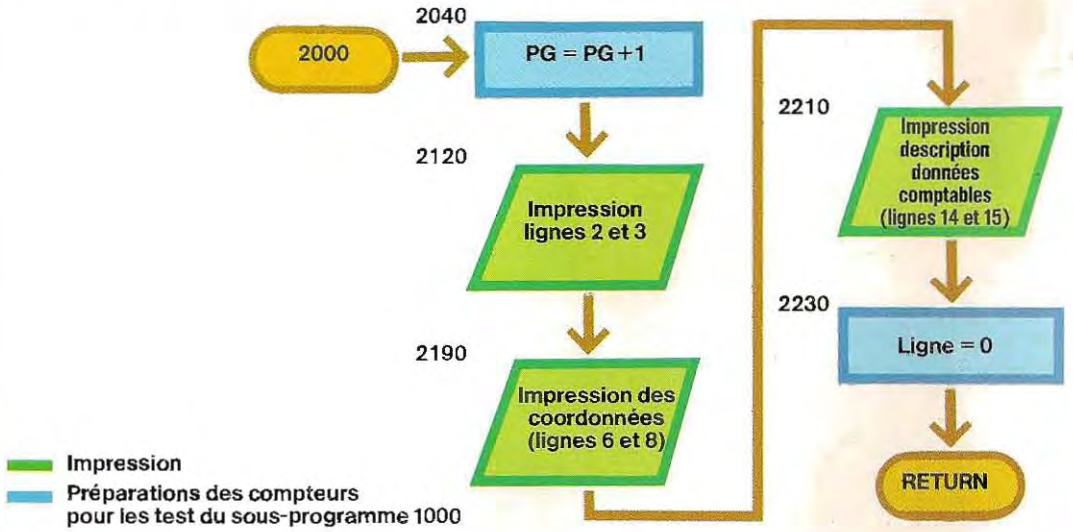
PROGRAMME PRINCIPAL D'IMPRESSION D'UN REGISTRE CLIENTS (sur 80 colonnes)



SOUS-PROGRAMME 1000 : GESTION DES IMPRESSIONS



SOUS-PROGRAMME 2000 : IMPRESSION EN-TETE ET COORDONNEES



PROGRAMME D'EDITION SUR 80 COLONNES

```
5 ' ** PROGRAMME D'EDITION SUR 80 COLONNES **
10 '
15 '              FICHIER = TA880C
20 '
25 ' VARIABLES UTILISEES
30 ' ** coordonnees clients
35 ' NOM DE CHAMP      SIGNIFICATION          TYPE           LONGUEUR
40 ' CC                CODE CLIENT            numerique       3
45 ' NO$               NOM                    alphabetique    13
50 ' RU$               RUE                    alphabetique    15
55 ' NU                NUMERO DE RUE          numerique       5
60 ' TL                TELEPHONE              numerique       12
65 ' CP                CODE POSTAL            numerique       5
70 ' VI$               VILLE                  alphabetique    15
75 ' DP$               DEPARTEMENT            alphabetique    15
80 '       * donnees comptables
85 ' NOP               NUMERO D'OPERATION     numerique       3
90 ' DT                DATE                   numerique       8
95 ' Z1                MONTANT                numerique       8
100 ' P                REMISE                 numerique       2
105 ' Z2               MONTANT NET            numerique       8
110 ' DS               DESCRIPTION            alphabetique    15
115 '
120 ' CC = CODE CLIENT EN COURS
125 ' CP = CODE CLIENT PRECEDENT
130 ' DI = DATE D'IMPRESSION
135 ' PG = NUMERO DE PAGE EN COURS
140 '
145 ' INITIALISATIONS
150 BI$=CHR$(27)+"+"+CHR$(7)    ' Efface l'ecran et emet un "bip"
151 CP=0            ' Initialisation du code client precedent
152 PG=0            ' Initialisation du numero de page
153 Z3=0            ' Initialisation de la somme des montants nets
155 '
160 ' ** PROGRAMME PRINCIPAL **
165 INPUT "DATE D'IMPRESSION ";DI
170 PRINT "INTRODUIRE LE CODE CLIENT (NUMERIQUE, 3 CHIFFRES MAXI)"
175 INPUT CC : IF CC=CP GOTO 215    ' Si le code client n'a pas change
180 INPUT "NOM ";NO$                ' on passe directement a la
185 INPUT "RUE ";RU$                ' saisie des donnees comptables
190 INPUT "NUMERO ";NU
```

```
195 INPUT "TELEPHONE ";TL
200 INPUT "VILLE ";VI$
205 INPUT "CODE POSTAL ";CP
210 INPUT "DEPARTEMENT ";DP$
215 INPUT "NUMERO D'OPERATION ";NOP
220 INPUT "DATE ";DT
225 INPUT "MONTANT ";Z1
230 INPUT "REMISE (%) ";P
235 INPUT "DESCRIPTION (15 CARACTERES MAX) ";DS$
240 GOSUB 1000              ' Calculs et impression
250 CP=CC
255 '
260 PRINT BI$
270 IF REP$<>"OUI" GOTO 170
280 END
285 '
290 ' ** SOUS-PROGRAMME D'IMPRESSION DU FICHIER CLIENTS **
1000 IF CC=CP GOTO 1200    ' Si CC=CP, il n'y a pas eu
1010                       ' de changement de code
1020 IF PG=0 THEN GOSUB 2000 : GOTO 1200
1030 LPRINT TAB(55); : LPRINT USING "£££££.££";Z3
1040 Z3=0
1050 LPRINT CHR$(12)       ' Saut de page
1060 PRINT "EST-CE LE DERNIER CLIENT ? (OUI/NON)
1070 INPUT REP$
1080 GOSUB 2000        ' Impression coordonnees client
1090 '
1100 '
1110 '
1120 ' Calcul du montant net total
1130 '
1200 Z2=Z1/100*(100-P)     ' On deduit de Z1 la remise P
1210 '                     ' pour obtenir le montant net Z2
1220 Z3=Z3+Z2              ' Cumul dans Z3 des montants nets
1230 '
1240 ' Impression d'une ligne de donnees comptables
1245 '
1250 LPRINT TAB(1);:LPRINT USING "£££";NOP;:LPRINT TAB(20);DT;
1260 LPRINT TAB(32);:LPRINT USING "£££££.££";Z1;
1270 LPRINT TAB(47);:LPRINT USING "££";P;
1280 LPRINT TAB(55);:LPRINT USUNG "£££££.££";Z2;LPRINT TAB(65);DS$
1290 LPRINT
1300 LIGNE = LIGNE+1
1310 IF LIGNE>=30 THEN LPRINT CHR$(12) : GOSUB 2000
1320 RETURN
1330 '
2000 ' ** SOUS-PROGRAMME D'IMPRESSION D'EN-TETE **
2010 '    ET DE COORDONNEES CLIENT
2020 '
2030 LPRINT
2040 PG=PG+1       ' Incrementation du numero de page
2050 LPRINT TAB(70);"PAGE";PG
2060 LPRINT TAB(25);"SITUATION CLIENTS AU ";TAB(47);DI
2070 LPRINT
2080 '
2080 ' Impression des coordonnees
2100 '
2110 LPRINT
2120 LPRINT TAB(3);CC;TAB(8);NO$;TAB(27);TL
2130 LPRINT
2140 LPRINT TAB(2);NU;TAB(8);RU$;TAB(25);CP;TAB(32);VI$;TAB(48);DP$
2150 LPRINT : LPRINT
2160 LPRINT : LPRINT
2170 LPRINT
2180 '
2190 LPRINT TAB(5);"OPERATION";TAB(19);"EN DATE DU";TAB(32);"MONTANT";
2200 LPRINT TAB(45);"REMISE";TAB(57);"NET";TAB(65);"DESCRIPTION"
2210 LPRINT TAB(8);"NUMERO";TAB(47);"%"
2220 LPRINT
2230 LIGNE=0
2240 RETURN
```

RESULTATS DU PROGRAMME TAB80C (PAGE 1)

```
                                                                    PAGE 1
          SITUATION CLIENTS AU 4/10/84

  1 MARTIN       (06)25-88-10

123 RUE ST-JACQUES 28000 CHARTRES     EURE-ET-LOIR


 OPERATION   EN DATE DU   MONTANT    REMISE     NET      DESCRIPTION
   NUMERO                             %

       936   10/10/83    1230.00       10     1107.00   MARCH. FRAGILE
       937   11/10/83    1450.00       15     1232.00   MARCH. VARIEE
       938   12/10/83    9000.00       10     7380.00   EXPLOSIFS
       939   13/10/83   11920.00       10    10728.00   PDTS CHIMIQUES
                                             20447.00
```

RESULTATS DU PROGRAMME TAB80C (PAGE 2)

```
                                                                    PAGE 2
          SITUATION CLIENTS AU 4/10/84

  2 LEBLANC      (06)48-48-00

64 RUE G.-CLEMENCEAU 91405 ORSAY      ESSONNE


 OPERATION   EN DATE DU   MONTANT    REMISE     NET      DESCRIPTION
   NUMERO                             %

       800   10/11/83    9700.00        5     9215.00   JEUX
                                             9215.00
```

Impressions paramétrées. Cette application, limitée à un cas très simple, est effectuée dans un but de démonstration. Elle permettra au lecteur de se familiariser avec certaines instructions Basic, et d'acquérir les notions fondamentales de la méthodologie*.
La page 568 schématise l'organigramme du programme principal synthétisant les fonc-

* Nous n'aborderons pas ici l'utilisation et la manipulation des masques d'édition

Solutions du test 16

1 / Le programme exécutant les fonctions demandées peut prendre la forme suivante :

```
10  OPTION BASE 1
20  DIM A(4,5)
25  ' Saisie des éléments de la première ligne
30  FOR I=1 TO 5
40  PRINT "Elément:";I
50  INPUT "Valeur";A(1,I)
60  NEXT I
70  ' Calcul des éléments de la deuxième ligne
80  FOR I=1 TO 5
90  A(2,I)=2*A(1,I)
100 NEXT I
110 ' Calcul des éléments
    de la ligne 3
120 FOR I=1 TO 5
130 A(3,I)=3*A(1,I)
140 NEXT I
150 ' Calcul des éléments
    de la ligne 4
160 FOR I=1 TO 5
170 A(4,I)=4*A(1,I)
180 NEXT I
200 END
```

Les trois boucles calculant les éléments des trois lignes 2, 3, 4 peuvent être réunies en une seule. Les instructions des lignes 70 à 180 sont alors remplacées par les suivantes :

```
70  FOR J=2 TO 4  ' Sélection du numéro de ligne
80  FOR I=1 TO 5
90  A(J,I)=J*A(J,I)  ' Calcul de l'élément en Je ligne et Ie colonne
100 NEXT I
110 NEXT J
```

---

2 / La routine se résume à deux boucles imbriquées, la première sélectionnant la ligne, la seconde sélectionnant la colonne. Selon l'indice que l'on fait varier en premier, on obtient l'impression par ligne ou par colonne. On choisira par exemple :

```
1000 FOR J=1 TO 4
1010 FOR I=1 TO 5
1020 PRINT A(J,I)
1030 NEXT I
1040 NEXT J
1050 RETURN
```

---

3 / Le programme répétera trois fois l'instruction A$=INPUT$(1), suivie du contrôle de numéricité à l'aide de la fonction VAL (le code ASCII du caractère doit être compris entre 48 et 57). Si la donnée saisie n'est pas numérique, on affiche A$, sinon le programme doit demander une nouvelle valeur de A$.

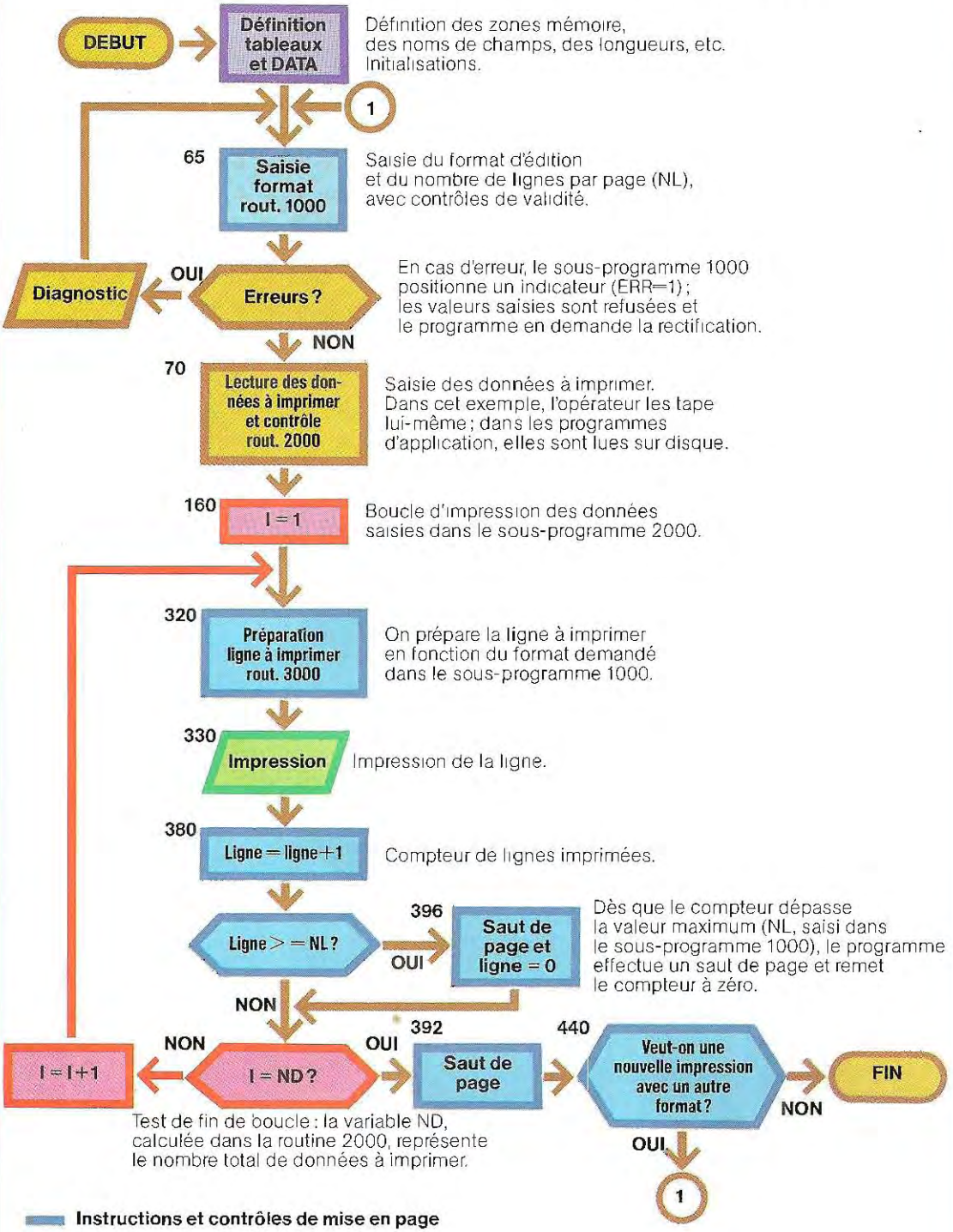
---

4 / On généralise ce programme en effectuant une boucle de 1 à N, réalisant les fonctions suivantes :

lecture d'un caractère [A$=INPUT$(1)] ;
contrôle de numéricité ;
affichage à l'écran si la donnée est acceptée.

Le programme est très semblable à l'exemple d'utilisation de la fonction INPUT$(N) présenté page 555.

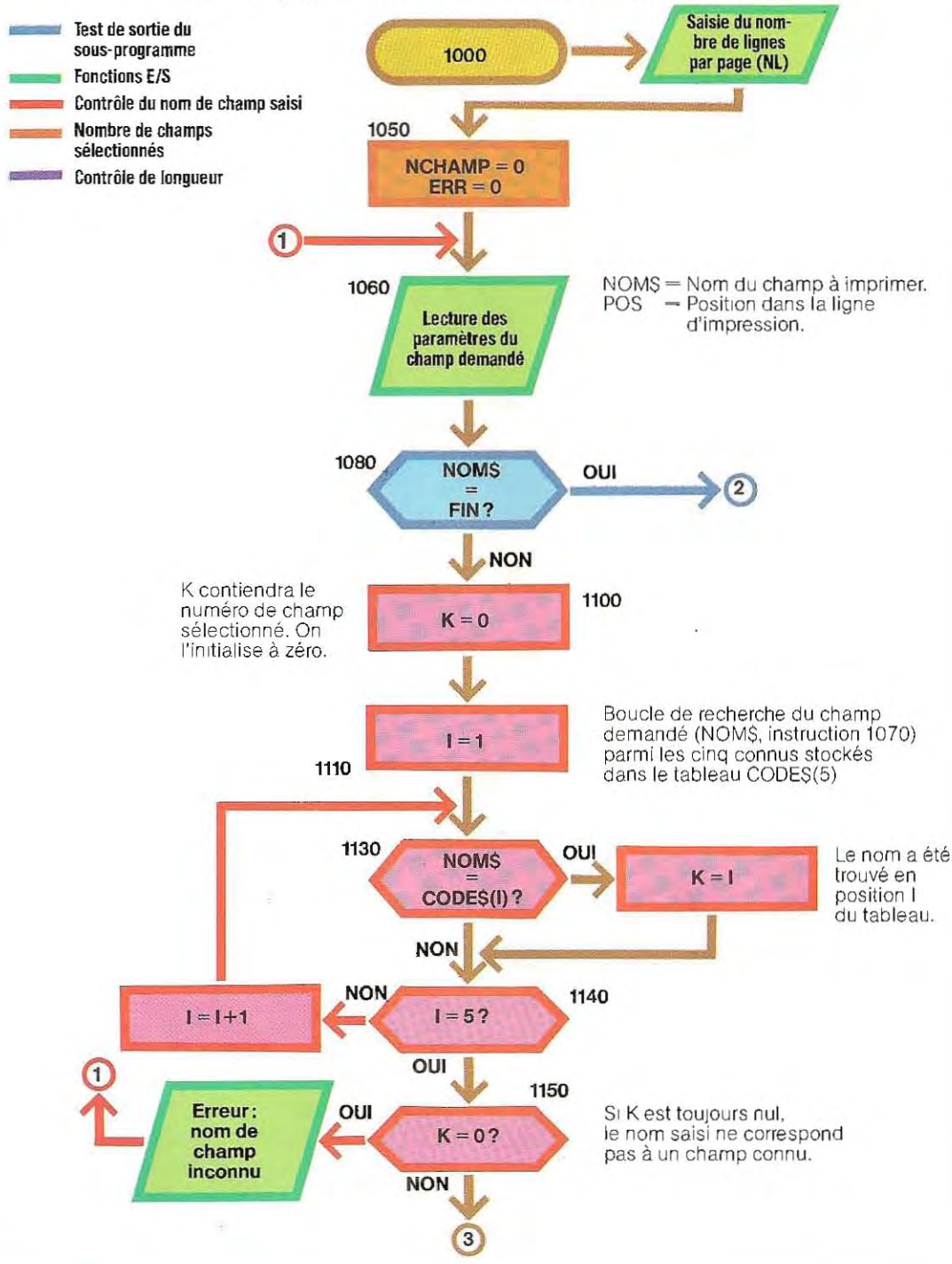
PROGRAMME PRINCIPAL : IMPRESSION PARAMETREE

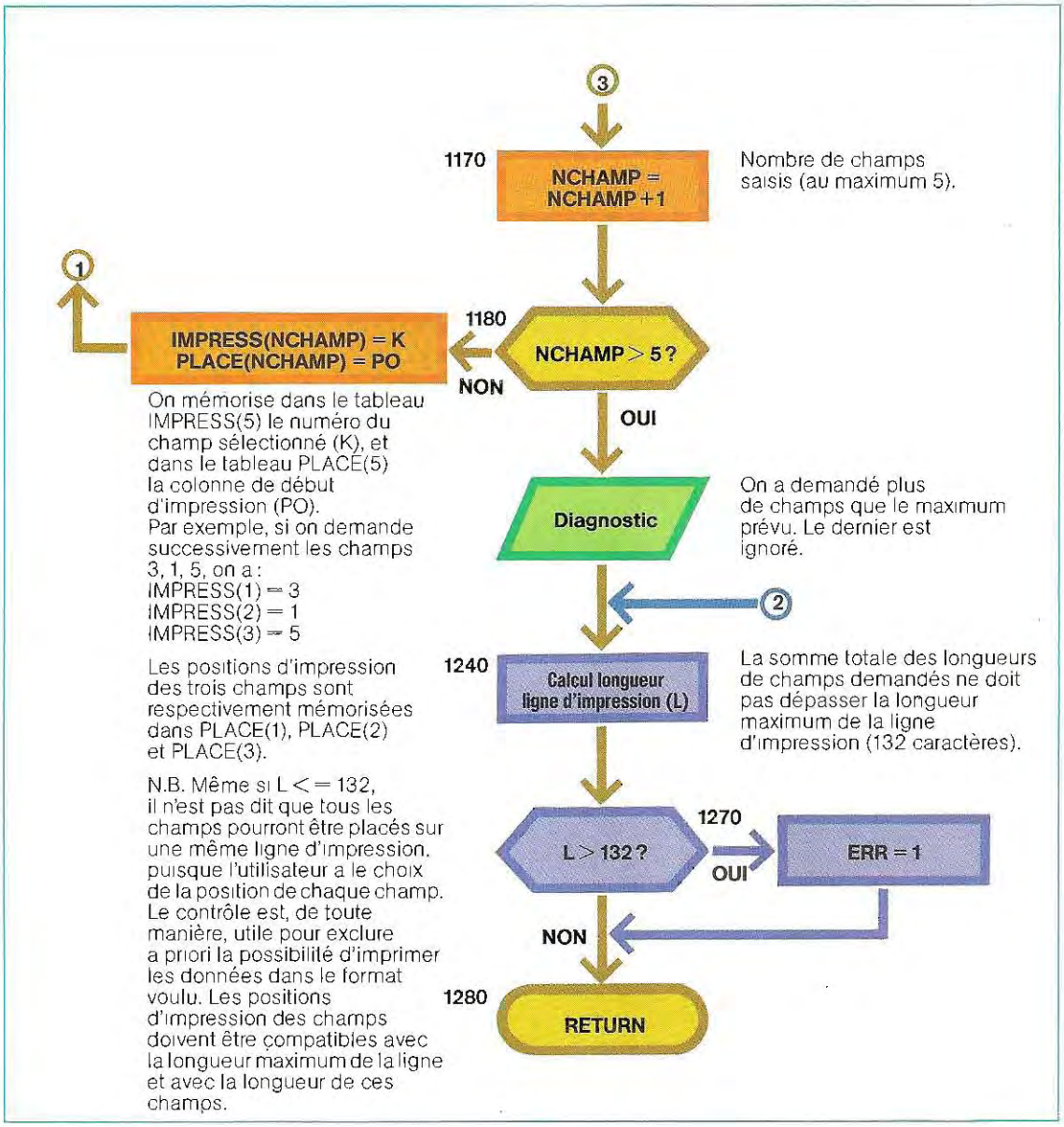


tions à exécuter ; les organigrammes suivants (pages 569 à 573) détaillent les routines 1000, 2000 et 3000 appelées par le programme principal. Le listing complet et certains résultats sont représentés pages 574 à 576.

Le programme principal définit un certain nombre de champs (cinq dans notre exemple), dont les noms et longueurs sont fixés une

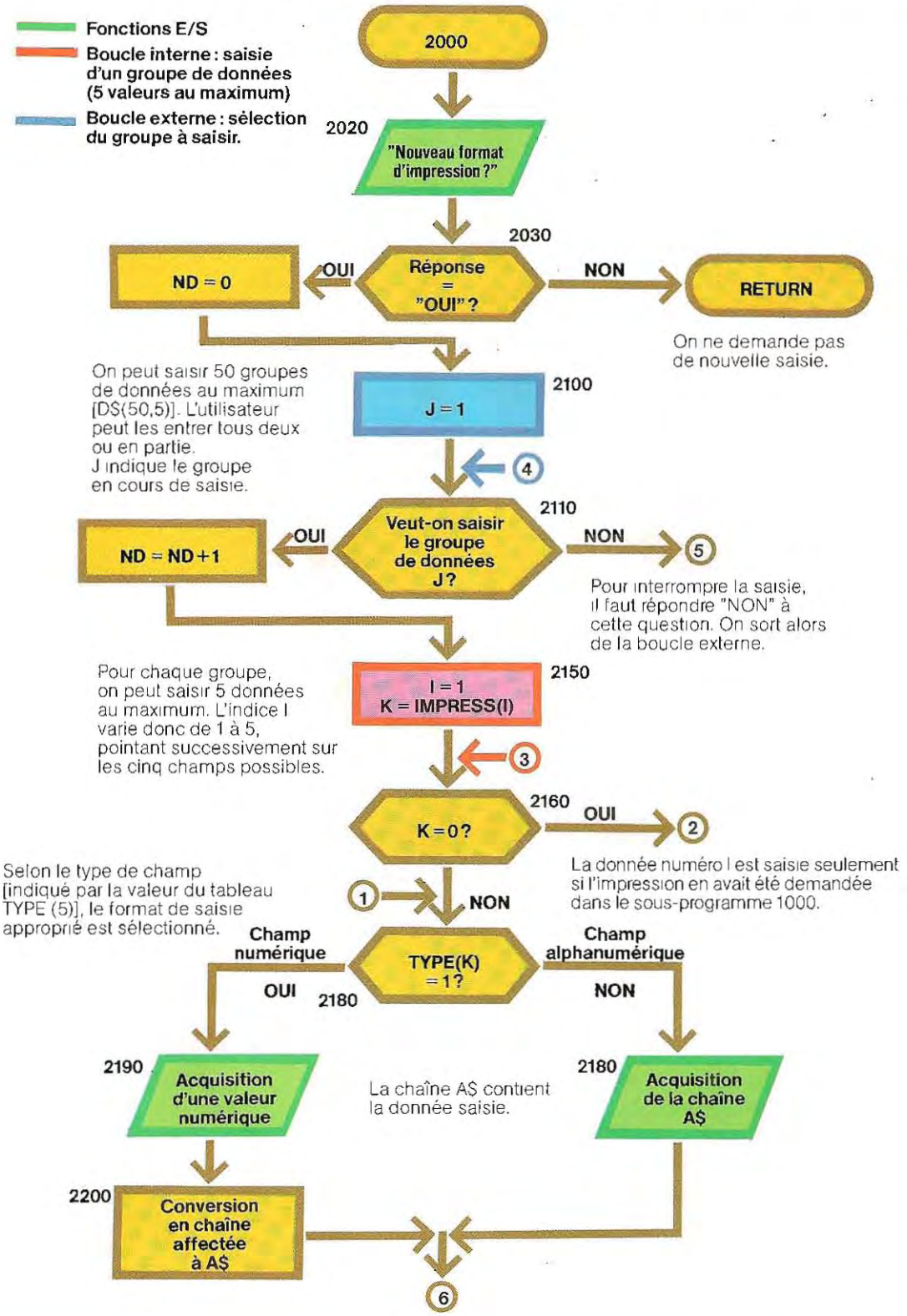
SOUS-PROGRAMME DE SAISIE DU FORMAT

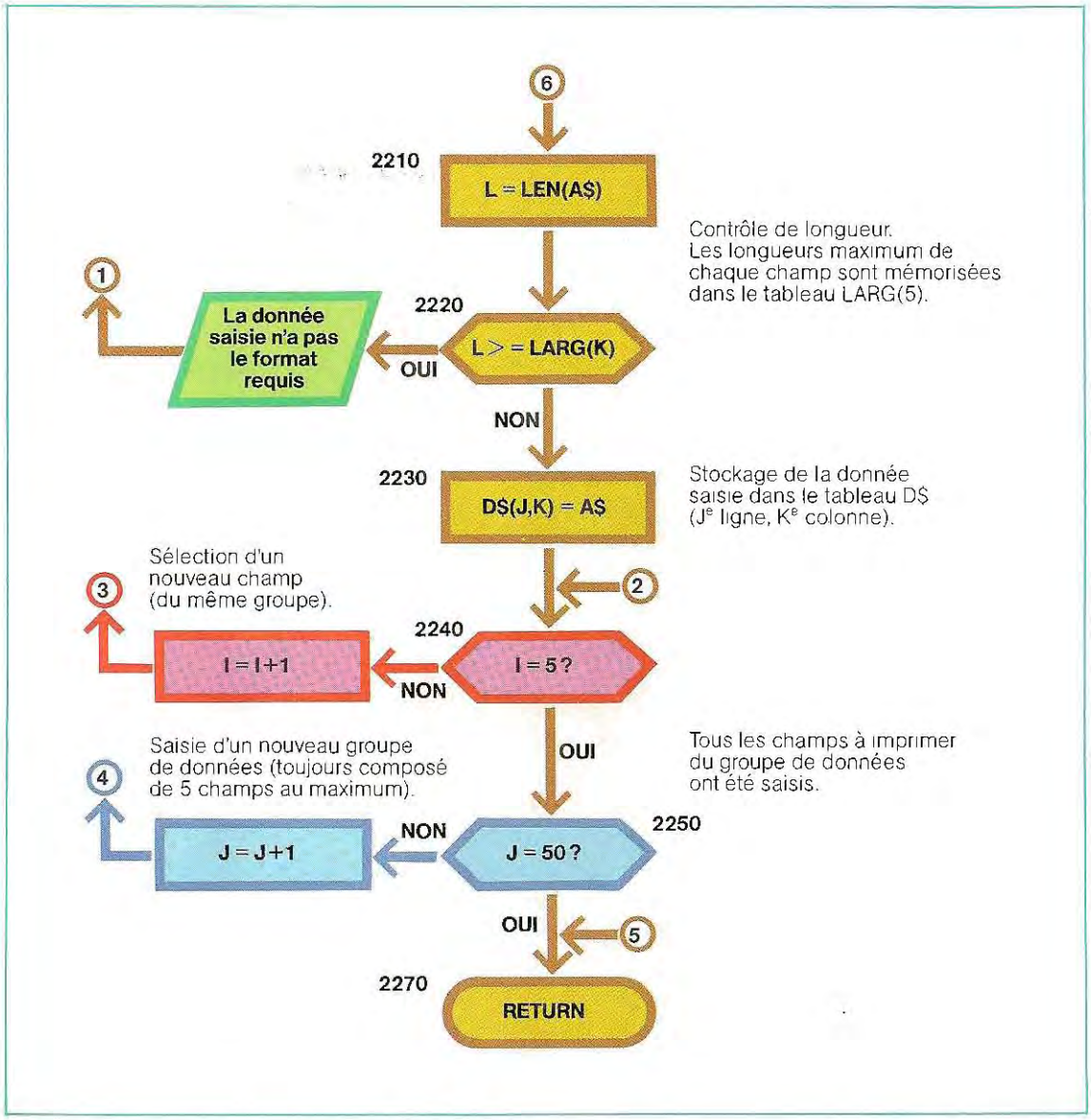




fois pour toutes (par exemple, lors d'une instruction DATA). Le sous-programme 1000 demande à l'utilisateur de spécifier successivement les champs qu'il désire traiter et la position où ils devront être imprimés.
Un certain nombre de contrôles sont prévus : vérification du nom de champ saisi, du nombre total de champs à imprimer, et de la longueur finale de la ligne d'impression (qui ne doit pas excéder 132 caractères). Pour ce dernier test, on cumule les longueurs respectives des différents champs.
La routine 2000 simule une lecture sur disque des groupes de données à imprimer (au maximum 50) : on demande à l'utilisateur de taper les donnés dont il a demandé l'impression dans le sous-programme 1000, et on contrôle la longueur de chaque chaîne saisie, qui sera ensuite stockée dans le tableau D$(50,5). La variable ND comptabilise le nombre total de groupes saisis.
Le sous-programme 3000 chargé de préparer chaque ligne d'impression (correspondant à un groupe de données), concatène successivement dans la chaîne A$ le contenu de chaque champ à imprimer (A2$, lu dans le tableau D$), et la chaîne d'espaces (A1$), qui séparera le champ en cours du précédent.

SOUS-PROGRAMME DE LECTURE ET DE CONTROLE DE DONNES A IMPRIMER



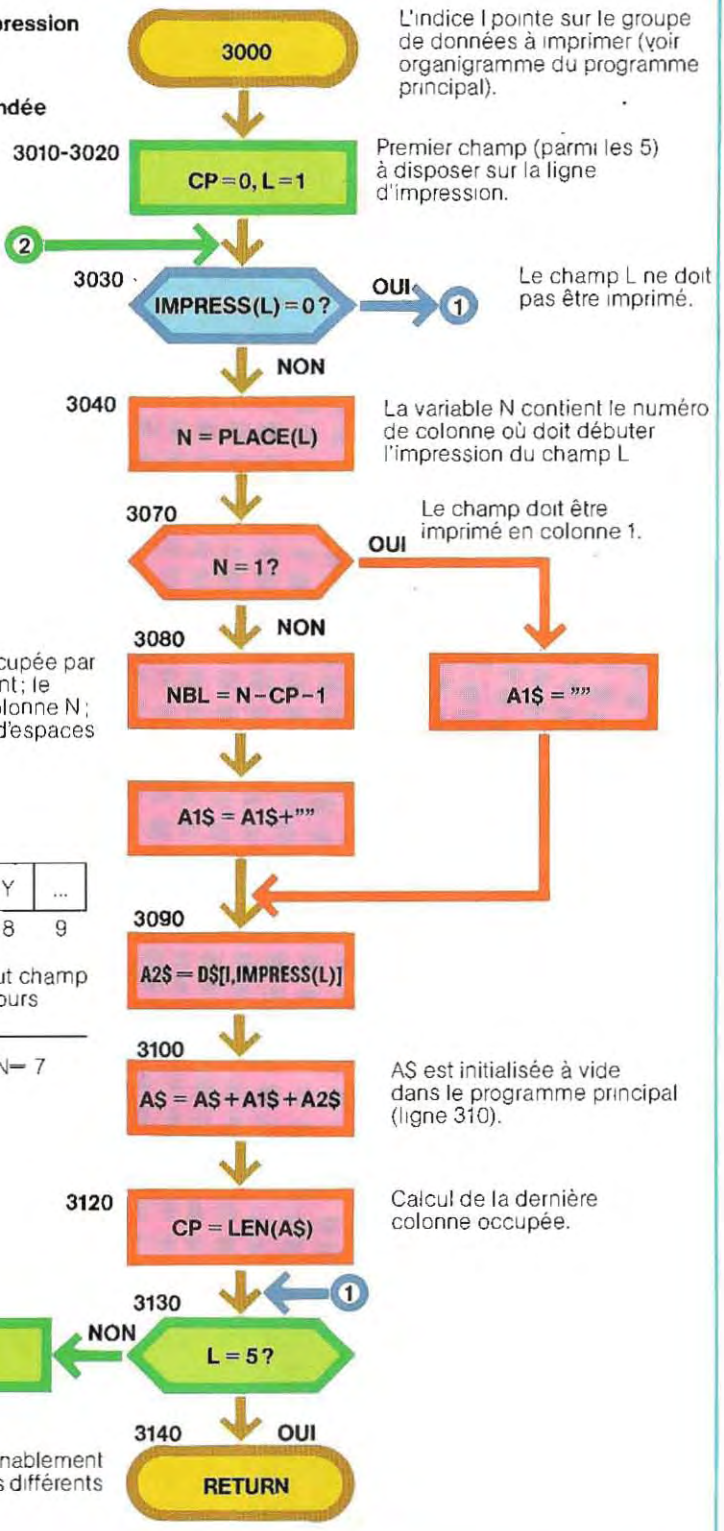


Notre programme d'application pourrait se compléter des fonctions réalisant par exemple des opérations arithmétiques de logiques entre les champs, ou d'autres plus sophistiquées (prévoyant une possibilité de représentation graphique). Avec un interface, permettant d'utiliser des valeurs stockées sur disque comme données d'entrée, on obtiendra un logiciel de portée très générale, offrant à l'utilisateur final un nombre varié de traitements spécifiques sans qu'il ait à entrer au cœur de la programmation.

Le déroulement de ce type de logiciel, s'il est destiné à une application pratique et non, comme dans notre exemple, à une démonstration, présente des difficultés notables et demande l'emploi de langages plus performants et plus proches de la machine que le Basic. Généralement, ces packages d'application sont écrits en Assembleur, ou même directement en langage machine sous forme de code hexadécimal. Cependant, l'utilisation de langages évolués tels que le Pascal ou le LISP se répand progressivement. Ces langages, employés en « intelligence artificielle » (surtout le LISP), permettent, grâce à leur haut niveau de structuration, la conception de programmes beaucoup plus complexes, et avec un gain de temps considérable par rapport à l'Assembleur.

SOUS-PROGRAMME DE PREPARATION DE LA LIGNE D'IMPRESSION

Boucle sur les champs d'impression
Préparation de la ligne
Saut des champs dont l'impression n'est pas demandée



CP est la dernière colonne occupée par la donnée du champ précédent ; le nouveau champ débute en colonne N ; il faut donc insérer un nombre d'espaces égal à NBL = N - CP - 1

Exemple

A	B	C				X	Y	...
1	2	3	4	5	6	7	8	9

Champ précédent	Nombre d'espaces à insérer	Début champ en cours
CP = 3	= N - CP - 1 = 7 - 3 - 1 = 3	N = 7

La chaîne A$ contient, convenablement séparés par des espaces, les différents champs à imprimer.

PROGRAMME D'IMPRESSION PARAMETREE

```
10 REM PROGRAMME D'IMPRESSION PARAMETREE
11 REM ********************************
20 REM
30 REM Executable sur APPLE II en Basic APPLESOFT
40 REM
45 REM
50 ERR=0
60 REM Variables
65 REM NL=nombre max de lignes/page (sous-pr. 1000)
70 REM ND=nombre max de donnees a imprimer (sous-pr. 2000)
75 REM
80 REM Tableaux
85 DIM CODE$(5): REM contient les 5 noms de champs prevus
90 DIM LARG(5): REM longueur des champs
95 DIM TYPE(5): REM type de champ (1=numerique, 2=alphanumerique)
100 DIM PLACE(5): REM numero de colonne de debut d'edition de champ
105 DIM IMPRESS(5): REM contient, dans l'ordre voulu, les pointeurs
106 REM                sur les champs a editer
110 DIM D$(50,5): REM tableau de travail contenant les donnees saisies
120 REM
125 REM Initialisations
130 BI$=CHR$(7): REM pour emettre un "bip"
131 SP$=CHR$(12): REM pour un saut de page
135 L$=CHR$(4)
140 REM
150 REM ** PROGRAMME PRINCIPAL **
155 REM
160 FOR I=1 TO 5
170 PRINT "CHAMP N.";I
180 INPUT "NOM DU CHAMP (3 caracteres) ";CODE$(I)
190 INPUT "LONGUEUR EN CARACTERES ";LARG(I)
200 INPUT "TYPE DE CHAMP (1=numerique 2=alphanumerique) ";TYPE(I)
210 PRINT L$;"PR#1"
215 REM Sur l'APPLE, cette instruction transfere le controle
216 REM a l'imprimante : tous les PRINT qui suivent
217 REM correspondront au LPRINT du Basic standard
220 PRINT "CHAMP N.";I; TAB(5);"NOM: ";CODE$(I); TAB(5);
222 PRINT "TYPE: ";TYPE(I); TAB(5);"LONGUEUR: ";LARG(I)
223 REM En Applesoft, l'instruction TAB(N) affiche N espaces,
224 REM au lieu de placer le curseur en position N
225 PRINT L$;"PR#0": REM rend le controle a la console
230 NEXT I
235 REM
240 GOSUB 1000
245 REM Saisie du format d'edition, du nombre de lignes/page,
246 REM validation des donnees saisies
247 REM
250 IF ERR=1 THEN PRINT BI$: PRINT "ERREUR: CORRIGEZ LES VALEURS"
260 IF ERR=1 THEN PRINT "ENTREES": GOTO 240
265 REM
270 GOSUB 2000
275 REM saisie des donnees, tapees au clavier par l'operateur
280 REM (dans les applications reelles, elles sont lues sur disque)
290 REM
295 REM
300 FOR I=1 TO ND: REM boucle d'impression par la routine 3000
310 A$=""
320 GOSUB 3000
325 REM
330 PRINT L$;PR#1": PRINT A$: REM l'imprimante edite la ligne
340 REM selon le format saisi en 1000
360 PRINT L$;"PR#0"
370 REM
380 LIGNE=LIGNE+1: REM compteur de lignes imprimees
```

```
390 IF LIGNE<NL GOTO 400
392 PRINT L$;"PR#1": PRINT SP$: REM saut de page
394 PRINT L$;"PR#0"
396 LIGNE=0
397 REM
400 NEXT I
410 REM
415 REM
420 PRINT L$;"PR#1": PRINT SP$: REM saut de page
425 PRINT L$;"PR#0"
430 PRINT BI$
435 REM
440 PRINT "VEUT-ON UNE IMPRESSION AVEC UN NOUVEAU FORMAT ? (OUI/NON)"
450 INPUT REP$
460 IF REP$="OUI" GOTO 160
470 REM
480 END : REM fin pr. principal
490 REM
500 REM
1000 REM ** SOUS-PROGRAMME DE SAISIE DU FORMAT **
1010 REM
1020 PRINT BI$
1030 PRINT "ENTREZ LE NOMBRE DE LIGNES PAR PAGE"
1040 INPUT NL
1045 REM
1050 NCHAMP=0: REM initialise NCHAMP (nombre a imprimer)
1060 REM lecture des parametres du champ
1070 INPUT "NOM DU CHAMP (tapez FIN pour sortir) ";NOM$
1080 IF NOM$="FIN" GOTO 1220
1090 REM
1100 K=0: REM K contiendra un pointeur vers le champ demande
1110 FOR I=1 TO 5: REM boucle de recherche de NOM$
1120 REM              parmi les champs connus (CODE$)
1130 IF NOM$=CODE$(I) THEN K=I: REM le nom se trouve en position I
1140 NEXT I
1145 REM
1150 IF K=0 THEN PRINT "ERREUR, NOM DE CHAMP INCONNU": GOTO 1070
1160 REM si K est encore nul, le nom saisi ne figure pas dans CODE$
1165 REM
1170 NCHAMP=NCHAMP+1
1180 IF NCHAMP>5 THEN PRINT "PLUS DE CHAMPS QUE PREVU": GOTO 1220
1185 REM
1190 IMPRESS(NCHAMP)=K:PLACE(NCHAMP)=PO
1200 GOTO 1070
1210 REM
1220 L=0
1230 FOR I=1 TO NCHAMP
1240 L=L+LARG(IMPRESS(I))
1250 NEXT I
1260 REM
1270 IF L>132 THEN ERR=1
1280 RETURN
1290 REM
1300 REM
2000 REM ** ROUTINE DE LECTURE ET CONTROLE DES DONNES A IMPRIMER **
2010 REM
2020 INPUT "SAISIE DE NOUVELLES DONNEES (OUI/NON) ";REP$
2030 IF REP$="NON" GOTO 2270
2040 PRINT BI$
2050 PRINT "on peut saisir au maximum 50 groupes de donnees"
2060 REM (car D$(50,5) c.f. ligne 110)
2070 PRINT "l'utilisateur peut les entrer tous ou en partie"
2080 PRINT
```

```
2090' ND=/
2100 FOR J=1 TO 50
2110 PRINT "SAISIE DU GROUPE DE DONNEES ";J;"(OUI/NON)?"
2120 INPUT REP$
2130 IF REP$="NON" GOTO 2270
2140 ND=ND+1
2150 FOR I=1 TO 5
2155 K=IMPRESS(I)
2160 IF K=0 GOTO 2240
2170 PRINT "ENTREE DU CHAMP N.";K
2180 IF TYPE(K)<>1 THEN INPUT "chaine ";A$: GOTO 2210
2190 INPUT "valeur ";IV
2200 A$=STR$(IV)
2210 L=LEN(A$)
2220 IF L>LARG(K) THEN PRINT "DONNEE SAISIE HORS FORMAT": GOTO 2180
2230 D$(J,K)=A$: REM memorisation de la donnee saisie
2240 NEXT I
2250 NEXT J
2260 REM
2270 RETURN
2280 REM
3000 REM ** SOUS-PROGRAMME PREPARANT LA LIGNE A IMPRIMER **
3010 CP=0
3020 FOR L=1 TO 5
3030 IF IMPRESS(L)=0 GOTO 3130
3040 N=PLACE(L): REM N=numero de colonne ou commencera
3050 REM          l'edition du champ L
3060 A1$=""
3070 IF N=1 GOTO 3090
3080 FOR NBL=1 TO N-CP-1:A1$=A1$+" ": NEXT NBL
3090 A2$=D$(I,IMPRESS(L))
3100 A$=A$+A1$+A2$
3110 REM calcul de la derniere colonne occupee
3120 CP=LEN(A$)
3130 NEXT L
3140 RETURN
```

```
CHAMP N.1   NOM : CLI   TYPE : 2   LONGUEUR : 20
CHAMP N.2   NOM : TEL   TYPE : 1   LONGUEUR : 9
CHAMP N.3   NOM : CPO   TYPE : 1   LONGUEUR : 5
CHAMP N.4   NOM : VIL   TYPE : 2   LONGUEUR : 15
CHAMP N.5   NOM : TOT   TYPE : 1   LONGUEUR : 8
    NOM DU CLIENT       VILLE RESIDENCE    1000
    UNTEL               PARIS              0
    MARTIN              DIJON              1572.12
    DURAND              LYON               -3256.5

CHAMP N.1   NOM : CLI   TYPE : 2   LONGUEUR : 3
CHAMP N.2   NOM : TEL   TYPE : 1   LONGUEUR : 9
CHAMP N.3   NOM : CPO   TYPE : 1   LONGUEUR : 5
CHAMP N.4   NOM : VIL   TYPE : 2   LONGUEUR : 3
CHAMP N.5   NOM : TOT   TYPE : 1   LONGUEUR : 10
     100    TELEPHONE       VIL       1000
     110    254-90-39       PAR       0
     120    54-90-39        DIJ       1572.12
     130    51-31-48        LYO       -3256.5

CHAMP N.1   NOM : CLI   TYPE : 2   LONGUEUR : 10
CHAMP N.2   NOM : TEL   TYPE : 1   LONGUEUR : 9
CHAMP N.3   NOM : CPO   TYPE : 1   LONGUEUR : 5
CHAMP N.4   NOM : VIL   TYPE : 2   LONGUEUR : 3
CHAMP N.5   NOM : TOT   TYPE : 1   LONGUEUR : 10
      NOM CLIENT        75000   VIL   10000
      LEBLANC           75018   PAR   0
      MARTIN            22000   DIJ   1572.12
      DURAND            69000   LYO   -3256.5
```

Gestion de l'imprimante

L'instruction LPRINT telle que nous l'avons utilisée jusqu'à présent donne accès à plusieurs formats d'impression. Cette gamme s'avère cependant insuffisante à la majorité des programmes d'application.
On peut, par exemple, être tenu de modifier la taille des caractères ou l'espacement entre deux lignes consécutives. Ces conditions d'impression particulières sont imposées à l'imprimante par des commandes spéciales, généralement un ou deux caractères, qui sont alors interprétés comme des ordres et non des chaînes à imprimer.
Les constructeurs insèrent toujours dans le manuel d'utilisation une liste des commandes et des codes correspondants. Ces codes de l'imprimante sont pour la plupart constitués de caractères spéciaux (CONTROLE ou ESCAPE). On est donc généralement obligé d'avoir recours au code ASCII de ces caractères par l'intermédiaire de la fonction BASIC CHR$(N). Ainsi, le caractère CTRL L ou FF (saut de page) correspond au nombre décimal 12 selon le code ASCII. On exécutera donc cette instruction en tapant :

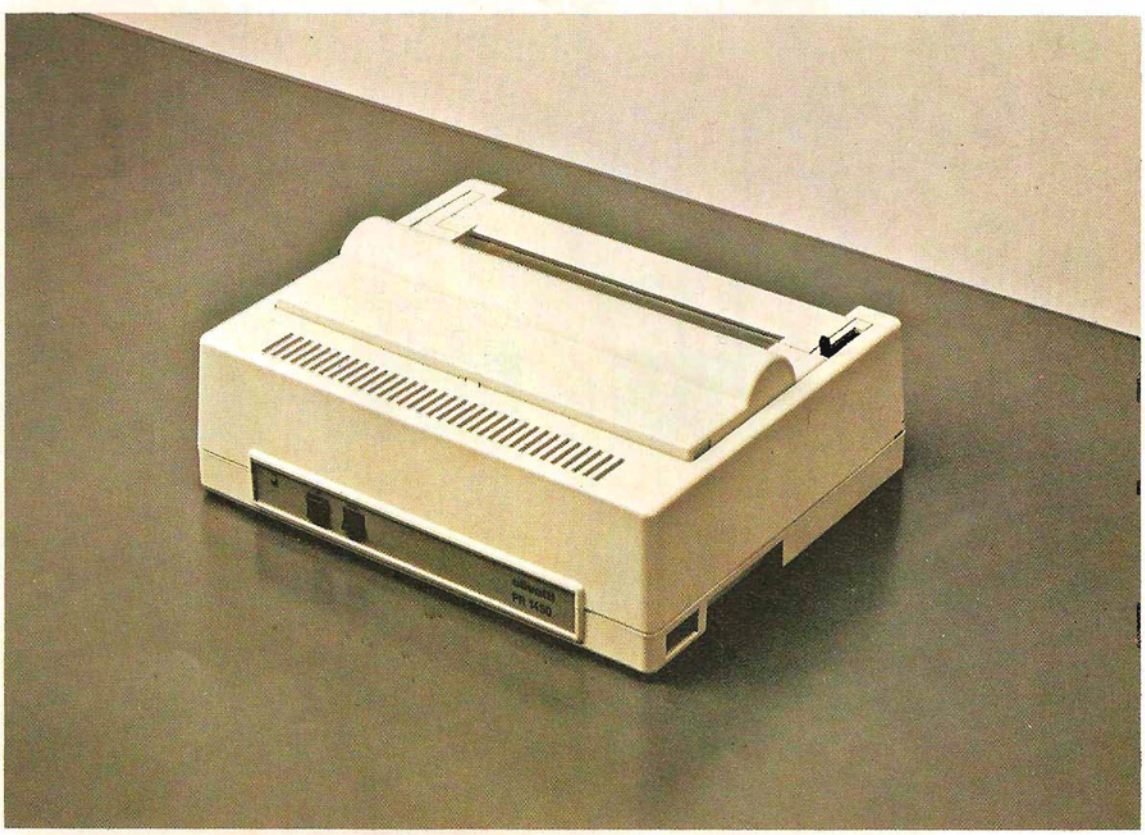
```
LPRINT CHR$(12)
```

Précisons que les commandes de l'imprimante ne font l'objet d'aucune standardisation et n'ont généralement pas les mêmes effets d'une machine à l'autre.
Les programmes cités plus loin en exemple ont été rédigés pour une imprimante EPSON RX 80, mais la mise en œuvre de codes différents reste comparable quel que soit le type de matériel.

Caractéristiques techniques des imprimantes à aiguilles

Avant d'aborder la programmation des fonctions d'édition, nous allons donner un aperçu du fonctionnement des imprimantes et, plus particulièrement, du système à aiguilles. Les imprimantes à marguerite ont en effet un jeu de caractères figé et non programmable.

Une imprimante à aiguilles : le modèle PR1450 d'Olivetti.



Olivetti

Quant aux modèles à laser ou à jet d'encre, ils sont trop récents et trop coûteux pour être encore très répandus sur les petits systèmes. On trouvera ci-dessous le schéma de principe d'une imprimante. Les données en provenance de l'ordinateur sont stockées dans une première mémoire tampon (optionnelle), le buffer de ligne, puis dans le buffer d'impression. Elles sont ensuite transmises au générateur de caractères, qui les codifie, et les dirige vers la tête d'impression.

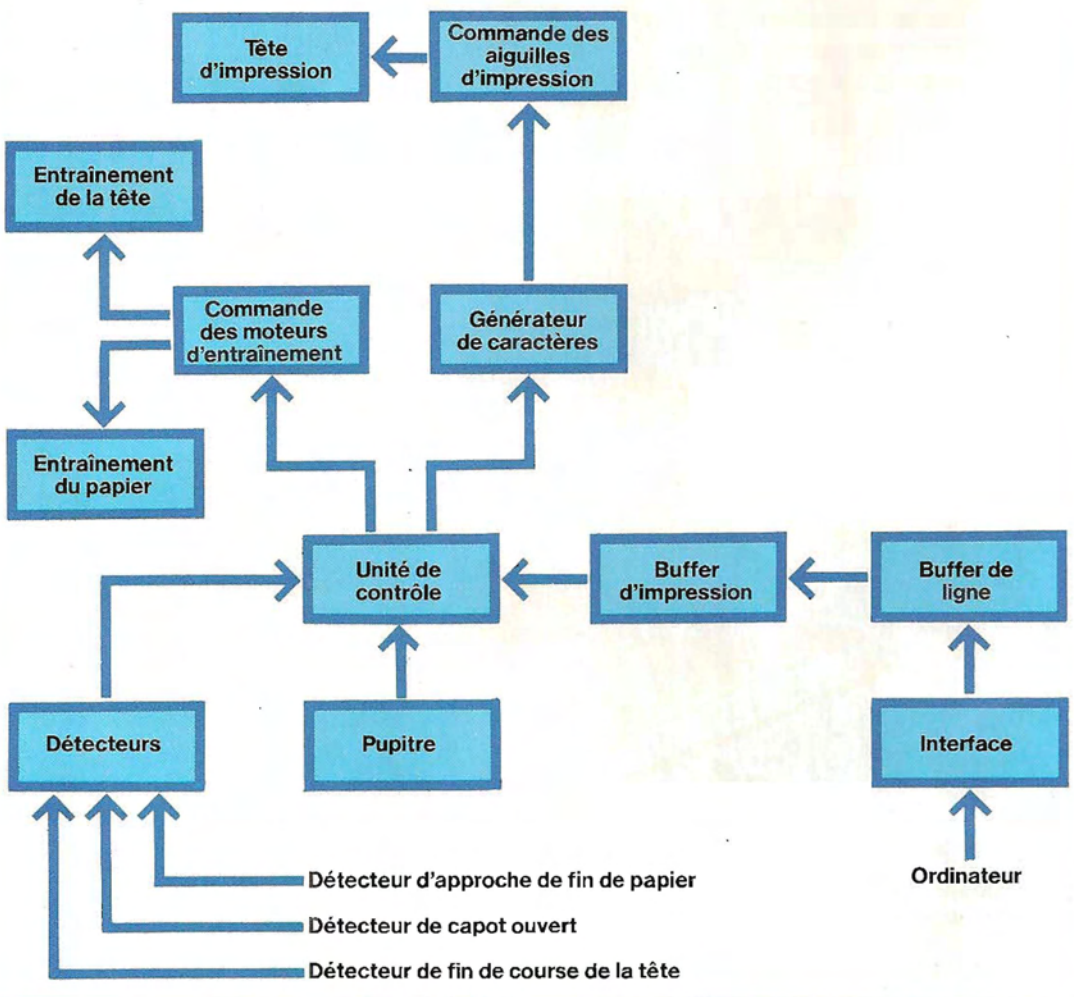
Nous allons maintenant décrire le fonctionnement d'une imprimante à aiguilles simple : l'Olivetti PR1450. L'impression est réalisée par l'impact d'une série d'aiguilles qui viennent percuter le ruban encreur et la feuille, les appliquant contre une réglette d'appui. Il en résulte une série de points dont la juxtaposition forme les caractères.

La tête d'impression regroupe neuf aiguilles. Elle est montée sur un chariot qui se déplace sur deux rails, parallèlement à la feuille de papier. Ce chariot est entraîné par un moteur à courant continu agissant sur une courroie crantée. Un levier permet de débloquer la réglette d'appui lors de l'introduction ou de la remise en place du papier.

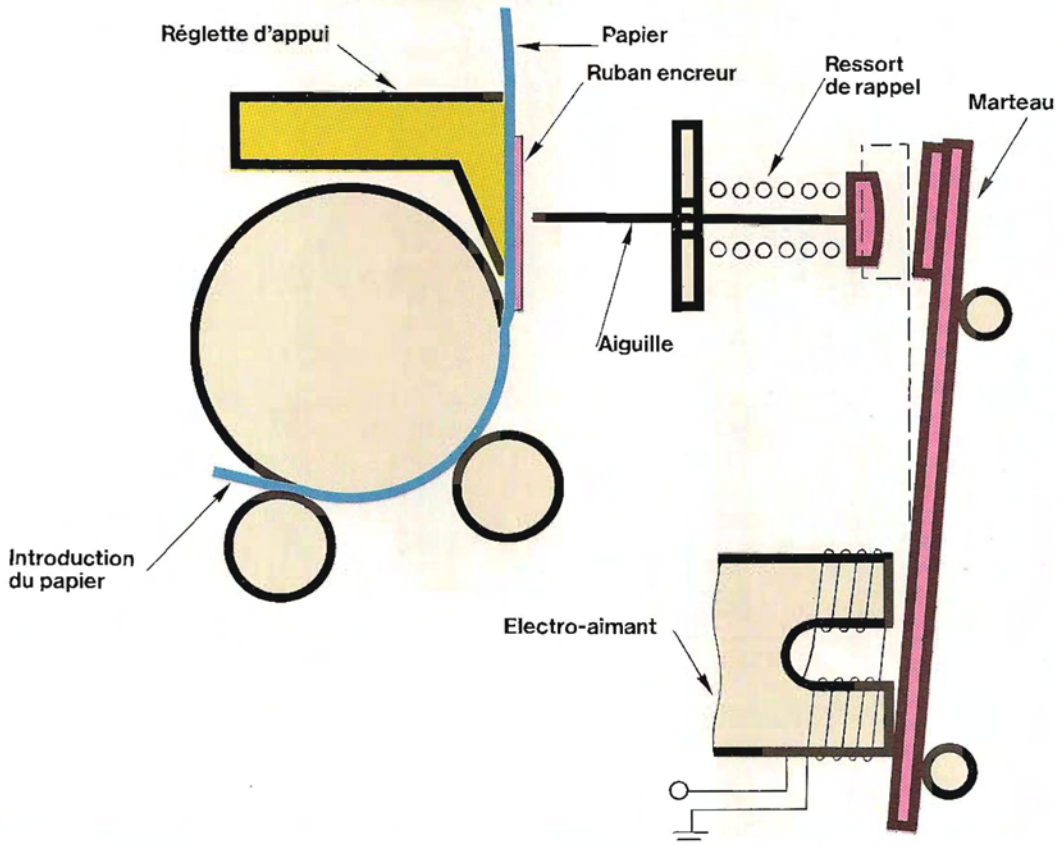
Les empreintes laissées par les aiguilles au cours d'un déplacement horizontal de la tête composent une ligne d'écriture.

L'impact sur les aiguilles est réalisé par des petits marteaux métalliques actionnés par des électro-aimants (bobines de fil conducteur qui créent un champ magnétique lors-

PRINCIPE DE FONCTIONNEMENT D'UNE IMPRIMANTE

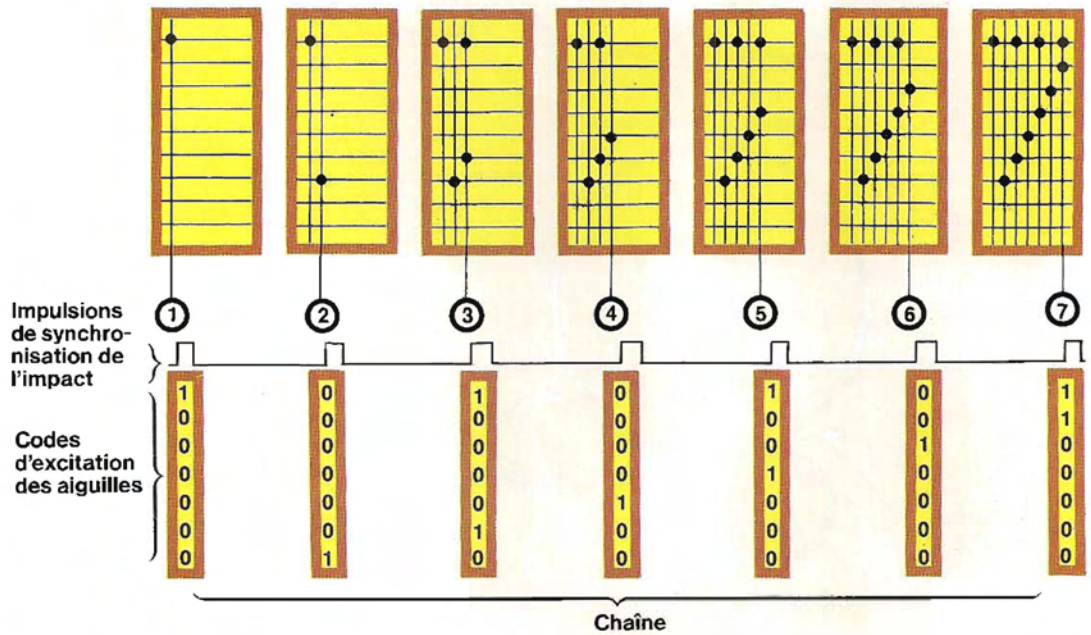


MECANISME D'IMPRESSION DU MODELE PR1450 OLIVETTI

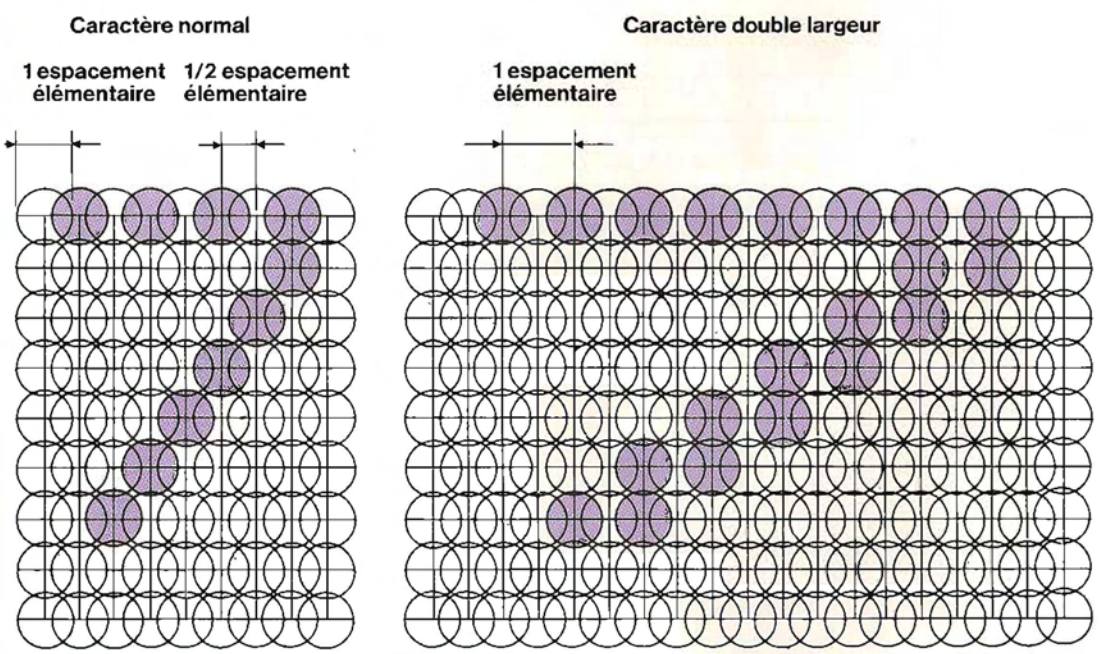


qu'elles sont alimentées par un courant électrique). Lorsqu'un courant est envoyé dans un de ces « solénoïdes », celui-ci attire le marteau qui, dans sa course, percute la tête de l'aiguille correspondante. Ce choc communique à l'aiguille assez d'énergie pour qu'elle poursuive sa course jusqu'à la feuille.
Elle rebondit ensuite et un ressort la ramène à sa position initiale (voir schéma ci-dessus). Il y a donc neuf électro-aimants dont l'excitation, durant le déplacement de la tête d'écriture le long de la ligne, provoque l'impression des combinaisons de points. Il va de soi que, pour former les caractères voulus, l'excitation de chacun doit suivre un ordre précis. Cette synchronisation est assurée par un programme spécialisé et par un générateur de caractères, transcodeur qui associe aux codes ASCII un certain nombre des points de la matrice de façon à former le caractère correspondant. On parle ainsi d'**impression matricielle**. La netteté d'un caractère est pro-

IMPRESSION DU CARACTERE 7 PAR L'IMPRIMANTE PR1450 OLIVETTI



COMPOSITION DES CARACTERES AVEC UNE MATRICE 9 × (4+3)



portionnelle au nombre de points dont il est constitué et il convient de trouver un compromis entre qualité d'impression (nombre de points de la matrice) et vitesse d'écriture en caractères par seconde.

La PR1450 utilise une matrice 9 × (4+3), c'est-à-dire une trame de neuf lignes et sept colonnes. La hauteur maximale d'un caractère est de neuf points (un par aiguille). Les sept colonnes sont en fait réparties en quatre prin-

cipales et trois intercalaires situées à un demi-espacement des principales. Il est impossible d'imprimer, sur la même ligne, deux points distants d'un demi-espacement. Les barres horizontales seront donc constituées de quatre points au maximum, les colonnes intermédiaires restant réservées aux barres obliques. On dispose donc d'une matrice de trente-six points parmi soixante-trois possibles (9 × 7).

Pour les caractères en double largeur, les points ne sont imprimés que sur les colonnes principales. Les huit codes d'excitation sont obtenus par un « et » logique entre le code correspondant et le code précédent du caractère normal.

Voici la liste des différents jeux de quatre-vingt-seize caractères disponibles sur l'imprimante PR1450 :

- INTERNATIONAL
- USASCII
- ALLEMAND
- FRANÇAIS
- ITALIEN
- ANGLAIS
- ESPAGNOL
- PORTUGAIS
- FINNO-SUEDOIS
- DANOIS-NORVEGIEN
- SUISSE

On sélectionne le jeu de caractères désiré soit par mini-interrupteurs (dip-switch), soit dans le programme lui-même.

De l'alphabet ASCII, seuls l'espacement et le caractère DEL (caractère d'oblitération) ne sont pas imprimés. Les caractères non prévus dans un jeu seront remplacés par le symbole « III ». On peut cependant « interdire » cette impression, mais aucun message ne sera alors émis. Sur la plupart des imprimantes, il est possible d'obtenir un format élargi des caractères. Avec l'Olivetti, on utilise pour cela la commande ESC 3 dont l'effet est immédiat. Cet ordre peut être introduit à n'importe quel endroit du texte et reste actif jusqu'à émission de la commande annulant ce mode (ESC 4). Cependant, l'impression redevient normale suite à toute modification de l'espacement et après une réinitialisation générale (Reset).

De même, à la mise sous tension, l'initialisation est automatique et l'impression ne passera en double largeur que si cela est demandé par le programme.

Exemple d'impression

Page 582, se trouve l'organigramme d'un programme qui illustre quelques-unes des possibilités offertes par l'imprimante EPSON RX 80.

Une liste des types d'impression disponibles

CAPACITES D'IMPRESSION D'UNE PR1450

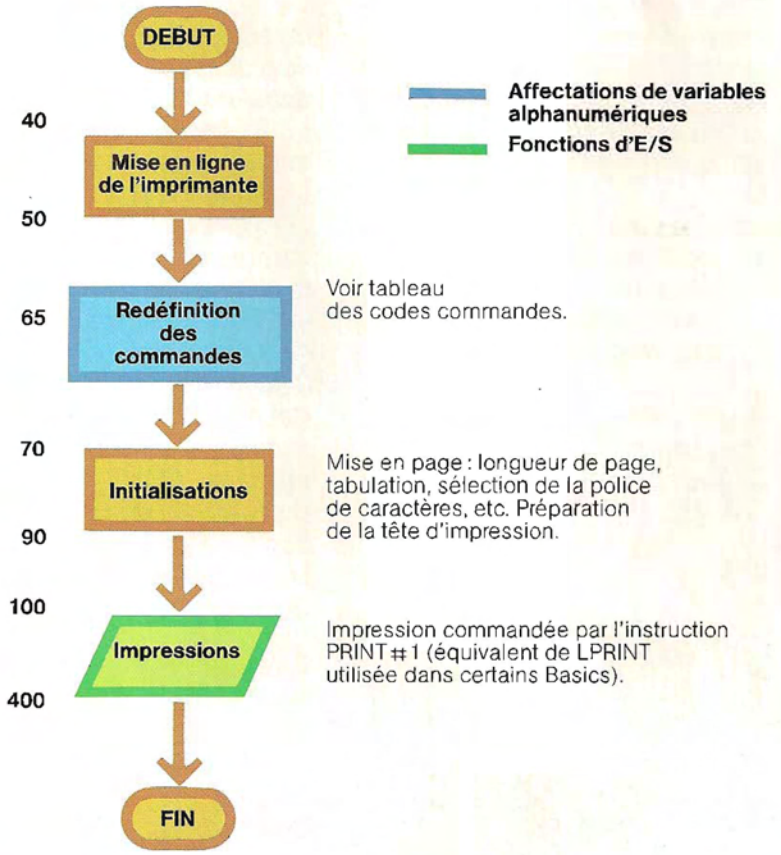
PRINTER CONTROL CODES

PRINT METHOD	Impact Dot Matrix
PRINTING SPEED	100 Char/Sec with fast carriage return
THROUGH PUT	50 lpm full line 86 lpm with 40 char
CHARACTERS COMPOSITION	9x7 (4+3) Dot Matrix
CHARACTERS PER LINE	1) 80 Char/line with 10 Char/inch 2) 132 Char/line with 16.6 Char/inch
LINE FEED	100 msec
COPIES	Up to 1 original and 2 copies

PRINTER CONTROL CODES

PRINT METHOD	Impact Dot Matrix
PRINTING SPEED	100 Char/Sec with fast carriage return
THROUGH PUT	50 lpm full line 86 lpm with 40 char
CHARACTERS COMPOSITION	9x7 (4+3) Dot Matrix
CHARACTERS PER LINE	1) 80 Char/line with 10 Char/inch 2) 132 Char/line with 16,6 Char/inch
LINE FEED	100 msec
COPIES	Up to 1 original and 2 copies

DEMONSTRATION DES TYPES D'IMPRESSION DE LA RX80 EPSON



sur cette imprimante (à l'exclusion des capacités graphiques) est fournie par le listing de la page 583.
Les caractères sont définis sur une matrice 9 × 9 en mode normal. Le programme provoque l'impression de la dénomination de chaque mode, du nombre maximal de caractères par ligne et de la densité correspondante en CPI (caractères par inch). Le recours fréquent aux codes de commande est dû à l'aspect démonstratif de ce programme, leur emploi étant infiniment plus restreint lors d'une application usuelle. Dans un but de simplification, ces commandes ont toutes été affectées à des variables alphanumériques, ce qui évite une utilisation systématique de l'instruction CHR$.

Tableau des codes de commandes utilisés.
A$ = ESC (changement de code).
C1$ = 50 : validation de l'écriture élonguée.
D1$ = DC4 : invalidation de l'écriture élonguée.
C2$ = SI : validation de l'écriture compressée.
D2$ = DC2 : invalidation de l'écriture compressée.
E$ = ESC E : validation de l'écriture en gras.
F$ = ESC F : invalidation de l'écriture en gras.
G$ = ESC G : validation du double passage.
H$ = ESC M : invalidation du double passage.
M$ = ESC M : validation de l'écriture ELITE.
P$ = ESC P : validation de l'écriture PICA.
T1$ = ESC – SOM : validation du soulignement.
T2$ = ESC – NUL : invalidation du soulignement.
SO$ = ESC S NUL : validation du mode exposant.
S1$ = ESC S SOM : validation du mode indice.
S$ = ESC T : invalidation de ces deux modes.
ESC 4 : validation de l'écriture en italique.
ESC 5 : invalidation de l'écriture en italique.
B$ = ESC 4 + "* ECRITURE "+ESC 5 : impression du mot ECRITURE en italique.
Ce tableau présente les différentes commandes disponibles sur l'EPSON RX80 ainsi que les codes de contrôle correspondants.

EXEMPLE DE TYPES D'IMPRESSIONS

```
10 TITLE"RXIMPMOD":CLEAR500,0:WIDTH20,50
20 PRINT"**********************  EPSON    RX-80  ** MODE  IMPRESSION *";
30 PRINT"**********************"+CHR$(11)+CHR$(23)
40 OPEN"O",#1,"COM0:MODE":WIDTH"COM0:",255
50 A$=CHR$(27):B$=A$+"4* ECRITURE "+A$+"5":C1$=CHR$(14):C2$=CHR$(15):D1$=CHR$(20
):D2$=CHR$(18)
60 E$=A$+"E":F$=A$+"F":H$=A$+"H":G$=A$+"G":P$=A$+"P":M$=A$+"M"
65 T1$=A$+"-"+CHR$(1):T2$=A$+"-"+CHR$(0):S0$=A$+"S"+CHR$(0):S1$=A$+"S"+CHR$(1):S
$=A$+"T"
70 DATA 27,64,27,67,0,11,27,108,6,27,82,1,13,10
80 RESTORE 70
90 READ A:PRINT#1,CHR$(A);:IFA<>10THEN90
100 PRINT#1;E$+G$+A$+CHR$(52)+"       IMPRIMANTE EPSON   RX
80"
110 PRINT#1,A$+CHR$(53)+"            MODE D'IMPRESSION"
120 PRINT#1,"                  * * * * * *":PRINT#1:PRI
NT#1,F$+H$
130 PRINT#1,B$+"NORMALE 80 caractères par lignes        (10 CPI)"
140 PRINT#1,E$+B$+"EN GRAS points doublés horizontalement  (10 CPI)"+F$
150 PRINT#1,G$+B$"DOUBLE PASSAGE décalés verticalement    (10 CPI)"
160 PRINT#1,E$+B$+"GRAS ET DOUBLE PASSAGE                  (10 CPI)"F$+H$
170 PRINT#1,C1$+B$+"ELONGUEE       (5 CPI)"
180 PRINT#1,C1$+E$+B$+"ELONGUEE-GRAS (5 CPI)"+F$
190 PRINT#1,C1$+G$+B$+"ELONGUEE ET   (5 CPI)"
200 PRINT#1,C1$+G$"        DOUBLE PASSAGE"
210 PRINT#1,C1$+E$+B$+"ELONGUEE-GRAS (5 CPI)"
220 PRINT#1,C1$+E$+G$"       ET DOUBLE PASSAGE"+F$+H$                    ex
230 PRINT#1,M$+B$+"ELITE   96 caractères par lignes                  (12 CPI)"
240 PRINT#1,G$+B$+"ELITE DOUBLE PASSAGE                              (12 CPI)"+
H$
250 PRINT#1,C1$+B$+"ELITE ELONGUEE     (6 CPI)"
260 PRINT#1,C1$+B$+"ELITE ELONGUEE ET  (6 CPI)"
270 PRINT#1,C1$+"       DOUBLE PASSAGE"+H$+P$
280 PRINT#1,C2$+B$+"COMPRESSEE 137 caractères par lignes
              (17 CPI)"
290 PRINT#1,G$+B$+"COMPRESSEE ET DOUBLE PASSAGE
              (17 CPI)"+H$
300 PRINT#1,C1$+B$+"COMPRESSEE ELONGUEE              (8.5 CPI)"
310 PRINT#1,C1$+G$+B$+"COMPRESSEE ELONGUEE ET          (8.5 CPI)"
320 PRINT#1,C1$+"       DOUBLE PASSAGE"+H$+D2$
330 PRINT#1,T1$+B$+"SOULIGNE       TECHNOLOGY RESOURCES S.A."+T2$
340 PRINT#1,B$+S0$+"EXPOSANT "+S1$+E$+"INDICE "+F$+S0$+M$+"EN ELITE ";
350 PRINT#1,S1$+P$+C2$+"COMPRESSEE"+S0$+D2$+C1$+"-ELONGUEE"+S$
360 PRINT#1,"     Y=AX"+S0$+"2"+S$+"+BX+C  avec  si B"+S0$+"2"+S$+"-4AC=0     X";
370 PRINT#1,S1$+"1"+S$+"=X"+S1$+"2"+S$+"=-B/2A"
380 PRINT#1,B$+"DE"+E$+G$+" PLUSIEURS MODES "+H$+F$+M$+"D'IMPRESSIONS "+P$;
390 PRINT#1,T1$+C2$+"SUR UNE "+C1$+"MEME "+D2$+"LIGNE"+T2$+CHR$(10)+CHR$(10)
```

```
* ECRITURE NORMALE 80 caractères par lignes        (10 CPI)
* ECRITURE EN GRAS points doublés horizontalement  (10 CPI)
* ECRITURE DOUBLE PASSAGE décalés verticalement    (10 CPI)
* ECRITURE GRAS ET DOUBLE PASSAGE                  (10 CPI)
* ECRITURE ELONGUEE                               (5 CPI)
* ECRITURE ELONGUEE-GRAS                          (5 CPI)
* ECRITURE ELONGUEE ET                            (5 CPI)
        DOUBLE PASSAGE
* ECRITURE ELONGUEE-GRAS                          (5 CPI)
        ET DOUBLE PASSAGE
* ECRITURE ELITE   96 caractères par lignes        (12 CPI)
* ECRITURE ELITE DOUBLE PASSAGE                    (12 CPI)
* ECRITURE ELITE ELONGUEE                         (6 CPI)
* ECRITURE ELITE ELONGUEE ET                      (6 CPI)
        DOUBLE PASSAGE
* ECRITURE COMPRESSEE 137 caractères par lignes    (17 CPI)
* ECRITURE COMPRESSEE ET DOUBLE PASSAGE            (17 CPI)
* ECRITURE COMPRESSEE ELONGUEE                    (8.5 CPI)
* ECRITURE COMPRESSEE ELONGUEE ET                 (8.5 CPI)
      DOUBLE PASSAGE
* ECRITURE SOULIGNE       TECHNOLOGY RESOURCES S.A.
* ECRITURE EXPOSANT INDICE EN ELITE COMPRESSEE -ELONGUEE
    Y=AX²+BX+C  avec  si B²-4AC=0    X₁=X₂=-B/2A
* ECRITURE DE PLUSIEURS MODES D'IMPRESSIONS SUR UNE MEME LIGNE
```

L'intelligence artificielle

Dans l'article qui suit, B. Meltzer, qui a travaillé sur un projet d'intelligence artificielle, présente quelques réflexions sur la similitude qu'on peut observer entre les processus cognitifs caractéristiques de l'esprit humain et les possibilités de l'ordinateur. Tout programme est en effet caractérisé par un certain degré de « savoir », dans la mesure où il rassemble une plus ou moins grande quantité de « cognitions » (connaissances) articulées, en fonction de la complexité des opérations qu'il doit exécuter.

L'énorme puissance de calcul de la dernière génération d'ordinateurs a permis de mettre au point des programmes extrêmement complexes et de plus en plus « cognitifs ».

On parle même d'**intelligence artificielle** pour les plus performants.

En intégrant son savoir dans ses programmes (rappelons que le savoir n'est pas seulement la possession de connaissances mais aussi la maîtrise du raisonnement), le programmeur y transfère, sous une représentation symbolique, une part de plus en plus importante de sa pensée.

En d'autres termes, on retrouve, dans les programmes, le matériel de la **psychologie cognitive**, sous une forme plus ou moins explicite.

Nombreux sont les psychologues, philosophes et autres chercheurs qui soutiennent aujourd'hui que, pour toute une série de raisons, il est impossible que des programmes informatiques représentent des structures mentales comparables à celles de l'esprit humain.

Je voudrais contredire ici quelques-unes de leurs objections.

Celle que l'on entend le plus fréquemment est que le fonctionnement du cerveau humain repose sur un support biologique de neurones qui n'ont rien à voir avec les transistors, les microcircuits, les films magnétiques ou autres éléments de l'ordinateur, et que, de ce fait, ses processus cognitifs ne peuvent qu'être totalement différents de ceux des ordinateurs.

Or, il me semble que bien peu d'informaticiens sont sensibles à cet argument dans la mesure où il est évident qu'un même programme tournant sur deux machines de structure différente effectue pourtant la même tâche de façon comparable.

On se heurte souvent à une autre objection concernant les logiciels proprement dits. Elle se fonde sur l'idée que les programmes cognitifs utilisent la vitesse de calcul de l'ordinateur et non l'intuition, l'imagination et l'inspiration qui sont celles du cerveau humain. Or, les exemples qui suivent sont la preuve éclatante que cette opinion est très éloignée de la réalité.

Il existe une profusion de programmes de jeux qui permettent aux hommes de se mesurer, avec plus ou moins de succès, à l'ordinateur. C'est ainsi qu'aux Etats-Unis, le tournoi d'échecs open du Minnesota a été remporté par le programme 4.5 mis au point par Slate et Atkin. Les programmes de jeu d'échecs les plus performants ne pratiquent pas l'analyse systématique de toutes les combinaisons possibles. Ils étudient, au contraire, certains enchaînements de coups, exactement comme le feraient des joueurs, chevronnés ou non.

Toutefois, l'exemple du backgammon (ou jacquet) est peut-être plus intéressant. En juillet 1979, à Montecarlo, l'italien Luigi Villa, champion du monde de backgammon, fut battu 7 contre 1 par le programme d'Hans Berliner, BKG 9.8 (Backgammon 9.8). Chercheur en intelligence artificielle à la Carnegie-Mellon University de Pittsburg, Berliner avait rédigé ce programme en développant ingénieusement le principe de l'un des premiers jeux informatisés, le jeu d'échecs de Samuel : les coups sont décidés au terme d'une analyse globale de la situation, à partir d'un nombre restreint d'éléments caractéristiques du jeu. Berliner a étudié le déroulement de son programme avec Villa et en a tiré les conclusions suivantes : « BKG 9.8 a incontestablement bien joué. La technique de Villa a été irréprochable pendant presque toute la partie alors que 8 fois sur 73, le programme n'a pas trouvé la meilleure réplique. BKG 9.8 n'a d'ailleurs été mis en difficulté que par l'une de ces erreurs, et, surtout, il s'est montré particulièrement brillant quand il s'est agi de faire preuve d'imagination. Et pourtant, sans même savoir ce que sont pions blancs et pions noirs, on aurait sans doute attribué à l'homme le jeu le plus sophistiqué par rapport à la machine ».

S'il est vrai que le langage est la « fenêtre de

l'esprit », de nombreux chercheurs ont tenté de mettre au point des programmes pour essayer de faire comprendre, traduire et résumer des textes en langage naturel par l'ordinateur. Voici un exemple qui permet d'apprécier les résultats obtenus jusqu'ici. Il s'agit d'un programme qui déchiffre les articles d'un journal, retient ce qui l'intéresse et rédige un bref résumé des faits dans la langue désirée (anglais, russe ou espagnol, par exemple).

Baptisé FRUMP (Fast Reading and Understanding Memory Program), ce programme a été conçu par l'unité de recherche sur la compréhension du langage naturel dirigée par Roger Schank à la Yale University.

L'article suivant, extrait d'un quotidien américain, a été communiqué à FRUMP : « Un violent tremblement de terre a secoué hier soir l'Italie septentrionale, détruisant des quartiers entiers de villes situées au nord-est de Venise, non loin de la frontière yougoslave ; d'après un communiqué du ministère de l'Intérieur italien, on compte au moins 95 morts et 1 000 blessés. Selon un porte-parole du gouvernement, on craint qu'il n'y ait au moins 200 morts enterrés sous les décombres, dans la seule ville d'Udine. Située sur le trajet de la principale liaison ferroviaire Rome-Vienne, la population de cette ville est de 90 000 habitants environ. Selon les gendarmes italiens, de graves dégâts se seraient produits dans une demi-douzaine de villes situées au pied des Alpes et des familles entières seraient ensevelies sous leurs maisons écroulées.

Dans de nombreuses localités, les communications n'ont pas encore pu être rétablies. L'intensité du séisme était de 6,3 sur l'échelle de Richter. Dans les régions habitées, une secousse de force 4 provoque généralement des dégâts modérés. La force 6 correspond à un grave séisme et 7 à une véritable catastrophe naturelle ».

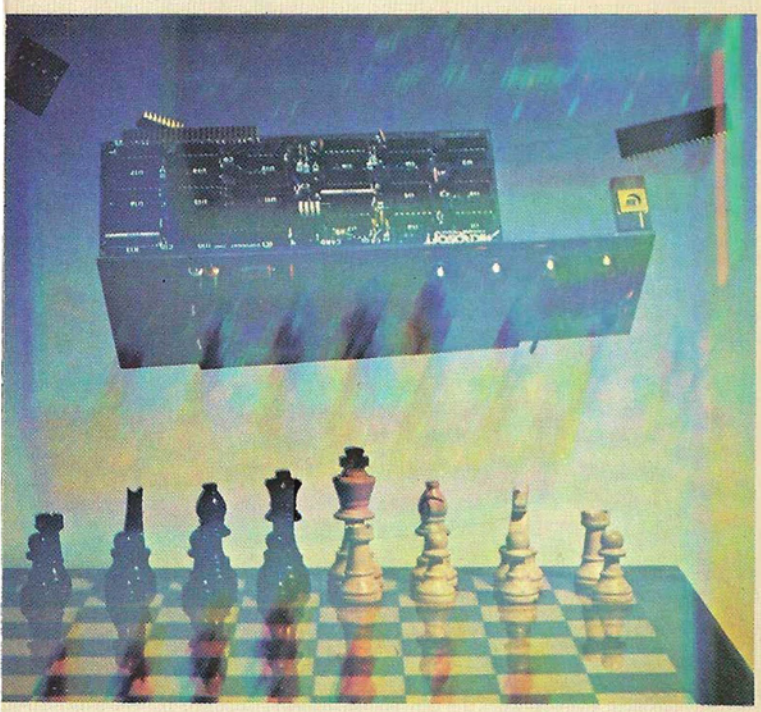
Voici le résumé que FRUMP a été capable de rédiger : « 95 personnes tuées et 1 000 blessées dans un tremblement de terre qui a frappé l'Italie. La secousse enregistrée est de force 6.3 sur l'échelle de Richter ».

L'équipe de Schank construit ses programmes autour de notions de la vie quotidienne. Nous allons maintenant parler d'un programme qui, au contraire, met en œuvre des concepts mathématiques et en découvre de nouveaux, sur lesquels il construit des « raisonnements » plus ou moins pertinents. Ce programme, appelé AM, a été écrit par un jeune américain, Douglas Lenat, qui l'a présenté lors de sa thèse soutenue en 1976 à l'Université de Stanford. Lenat s'est surtout attaché à rechercher des concepts et des modèles mathématiques dignes d'intérêt. En effet, AM n'est pas conçu pour démontrer des théorèmes, mais pour exécuter une série de tâches par ordre d'intérêt, en les abandonnant dès que la puissance de calcul requise se révèle insuffisante.

AM ne « connaissait » pas les notions de base sur les entiers naturels ; il a pourtant redécouvert tout seul de nombreux concepts, même si les notions de fractions et de nombres réels lui ont échappé.

Au départ, la base des connaissances d'AM se composait environ de 100 concepts et de 250 règles heuristiques qui lui permettaient de déterminer la démarche à entreprendre, en fonction du contexte. Elle incluait notamment les notions d'ensemble, de liste (AM est rédigé dans le langage LISP, particulièrement adapté au traitement des listes), de table de vérité, de relations d'appartenance et d'égalité, d'intersection, d'inversion et de composition. Lenat s'appuyait sur les travaux de psychologie infantile de Piaget et de Copeland pour affirmer que ce « savoir » de base correspondait aux connaissances d'un enfant de quatre ans. On peut discuter cette idée ; toujours est-il que le programme trouva rapidement, de façon empirique, qu'il n'existe pas de nombre sans diviseur et qu'un seul n'en possède qu'un. Il découvrit également que certains nombres n'admettent que deux diviseurs, eux-mêmes et l'unité. Il ajouta ainsi à sa base de départ un nouveau concept, celui de nombre premier.

Il faut également signaler que les recherches d'AM aboutirent à la découverte d'un nouveau concept dans la théorie des nombres, inconnu de Lenat lui-même mais qu'un jeune Indien de génie nommé Ramajuan avait étudié au début du siècle. AM découvrit les « nombres les plus composés », qui sont en quelque sorte le contraire des nombres premiers puisqu'ils admettent plus de diviseurs que les nombres qui leur sont inférieurs. Les premiers nombres de ce type sont 1 qui a un diviseur, 2 qui en a deux, 4 et 6 qui sont respectivement les plus petits nombres à avoir



Mallio Falcioni

Les programmes de jeux d'échecs sont parmi les applications les plus révélatrices de l'intelligence de l'ordinateur.

trois et quatre diviseurs, et ainsi de suite. En outre, AM parvint à redémontrer le théorème de la décomposition unique selon lequel tout entier naturel non premier se décompose de façon unique en un produit de facteurs premiers. Enfin, il découvrit l'énoncé de Goldbach qui dit que tout nombre pair est la somme de deux nombres premiers, et que l'on n'a pas encore pu démontrer.

A côté de ces succès remarquables, AM a essuyé des échecs qui sont tout aussi intéressants. J'ai déjà mentionné le fait qu'il fut incapable de découvrir les fractions. Il ne fit pas non plus de grands progrès dans la théorie des ensembles, excepté la découverte des relations les plus simples comme les lois de De Morgan. Il suivit souvent de mauvaises pistes en s'attardant, par exemple, sur l'idée de nombres décomposables de façon unique en une somme de deux nombres premiers. Lenat estimait dans sa thèse que 25 des concepts découverts par AM étaient justes (« bons »), 100 acceptables et les 60 autres erronés (« mauvais »).

Ces trois programme (jeu de backgammon, compréhension du langage naturel et découverte de propriétés mathématiques) ne sont que trois exemples assez récents de ce qu'on peut faire avec un ordinateur. Je les ai présentés pour étayer mon affirmation selon laquelle les procédures mises en œuvre dans les programmes les plus évolués ne sont pas purement « mécaniques », au mauvais sens du terme, mais reproduisent, dans une certaine mesure, les qualités de créativité, d'intuition et d'intelligence caractéristiques de l'esprit humain. On trouverait sans peine d'autres exemples tout aussi convaincants dans les domaines du diagnostic médical, de la perception visuelle, des tests de QI, etc.

L'informatique nous a habitués à utiliser des langages dont le but n'est pas de communiquer des informations mais de les représenter et de les traiter. C'est le cas des langages d'assemblage et des langages machine. D'après une analyse audacieuse proposée il y a peu de temps par le philosophe Aaron Sloman, ce serait même là la fonction première du langage, la fonction de communication n'étant qu'un « sous-produit ». Cela revient à dire que les premiers hommes auraient déjà possédé des langages « intérieurs » de communication et de représentation, qui auraient évolué en signes vocaux et imagés.

Pourquoi alors – si l'on admet l'existence de langages intérieurs se réduisant à une activité neuronique – ne pas établir une nette distinction entre ces langages intérieurs et les langages que nous parlons, comme le français ou l'anglais, et que j'appellerai, pour l'instant, les langages « extérieurs » ? Je crois cependant que le problème gagne en clarté à être abordé comme un seul phénomène.

Le processus de l'apprentissage humain – notamment lorsqu'il implique le maniement de symboles assez formels – constitue un solide argument en faveur d'une étroite continuité entre les deux types de langage. Prenons l'exemple des changements qui interviennent chez les enfants quand ils commencent à apprendre l'arithmétique. Lorsque leur maître leur explique l'addition pour la première fois, ils l'exécutent en suivant ses indications et en utilisant consciemment des mots et des phrases du langage naturel comme « je retiens » ou « 3 plus 3 égal 6 ». Mais quand ils savent faire les additions, ils les effectuent automatiquement, sans réfléchir, et il ne subsiste pratiquement plus rien dans

leur conscience. Pourtant, même au stade inconscient, il est évident que cette activité implique toujours une élaboration symbolique : simplement celle-ci passe désormais par le langage intérieur.

Il faut souligner combien la transition entre les deux langages se fait aisément et « naturellement ». Ainsi, face à une addition particulièrement longue et difficile, il nous arrive, alors même que nous avons bien intégré les mécanismes de calcul, d'effectuer la transition inverse et de revenir au langage extérieur en nous disant clairement : « il ne faut pas que j'oublie d'ajouter la retenue au total de cette colonne ».

Cela m'amène à formuler l'hypothèse suivante : « les processus mentaux de tout système biologique ne sont que des transformations qui s'opèrent au sein du langage propre à ce système, langage qui peut être sollicité à n'importe quel niveau ».

Cette notion de niveau nous est familière pour ce qui est des systèmes de traitement de l'information. Un programme source écrit dans un langage évolué ne peut être exécuté qu'après avoir été traduit d'abord en langage d'assemblage, puis en langage machine. Cette structure hiérarchisée constitue le schéma normal d'exécution des programmes. Toutefois, des variations peuvent modifier ce schéma.

Ainsi, sur certains systèmes, on a la possibilité d'insérer des passages en langage machine dans un programme écrit en langage évolué, afin de le rendre plus performant. A partir du moment où l'on fait ce genre de choses, la notion des différents niveaux de langage de la machine commence à s'estomper et il vaut mieux alors considérer que le programme est écrit dans un unique langage, constitué du langage source, de l'assembleur, du code machine ainsi que de toutes leurs variantes.

On peut faire un rapprochement entre ce langage et le langage unique que j'ai imaginé pour les êtres vivants.

J'ai évoqué les différents « niveaux » de mise en œuvre de ce système linguistique complexe : il faut bien trouver une explication au fait que nous puissions imaginer des objets sans les voir, rien qu'en les entendant nommer. Cela devient possible si l'on admet que le fonctionnement de nos différentes facultés cognitives repose sur un langage commun et qu'on peut donc accéder aux mêmes représentations en empruntant différents parcours linguistiques.

La poésie constitue un cas particulièrement intéressant car, tout en utilisant exclusivement le langage naturel, elle semble chercher à atteindre des couches plus profondes de la partie intérieure du langage. Elle éveille des résonances que l'usage « normal » du langage ne semble pas en mesure de susciter. Selon moi, la poésie s'emploie à recréer, à refléter et à représenter, par le langage naturel, des aspects extrêmement signifiants de notre langage intérieur.

La musique n'utilise pas non plus le discours habituel. Elle n'agit pas par son canal mais s'adresse plus directement à notre langage intérieur, ce qui lui confère une puissance d'évocation plus forte.

Si l'on considère les arts figuratifs, mon hypothèse semble impliquer que nos ancêtres reproduisirent sur les parois de leurs cavernes certains éléments de leur représentation intérieure de l'univers. Peut-être est-ce là ce qu'ont fait, après eux, tous les artistes ? A cet égard, il est révélateur que, dans ses travaux sur les modèles mathématiques de la perception visuelle des objets, Davis Marr ait défini, parmi les différents stades d'élaboration, celui des « stick figures », dont il désigne le graphisme caractéristique de l'une des périodes de Picasso.

Si, comme nous l'avons vu, les méthodes employées en intelligence artificielle sont bien adaptées à la représentation d'objectifs, de suppositions et d'interprétations, il ne devrait pas être très difficile de reconstituer des modèles satisfaisants pour les émotions. Le travail de pionnier entrepris par Freud sur les processus inconscients qui interviennent dans les rêves, les mots d'esprit et les lapsus ne semble pas avoir eu les répercussions scientifiques qu'on aurait pu attendre.

Les transformations qui interviennent dans les rêves ou dans les jeux de mots pourraient trouver une place, sous une formulation tout à fait rigoureuse, au sein des transformations informatiques déjà connues telles que l'interprétation ou la compilation. On devrait alors pouvoir écrire sans trop de peine un programme qui, à partir d'une certaine somme de connaissances, saurait nous raconter des histoires drôles avec beaucoup d'humour.

Présentation des diagrammes à barres

Lors de la préparation d'un travail quel qu'il soit, scientifique ou de gestion, il est souvent appréciable de disposer d'une représentation graphique des données. Un examen approfondi et des calculs précis s'appuient sur les tableaux de chiffres qui abondent dans rapports et comptes rendus. En revanche, pour visualiser l'aspect général d'une situation ou mettre en relief les traits caractéristiques d'un phénomène, il est préférable de transcrire ces tableaux en graphiques. Nous reviendrons plus en détail sur ces techniques, mais voyons dès à présent comment présenter des histogrammes par l'intermédiaire de l'imprimante.

Un **histogramme** est un type de représentation graphique souvent employé pour illustrer les tableaux simples. Supposons que l'on veuille rendre compte du chiffre d'affaires réalisé par une équipe de vendeurs. On pourra reporter dans un tableau le nombre d'articles vendus par chacun d'eux :

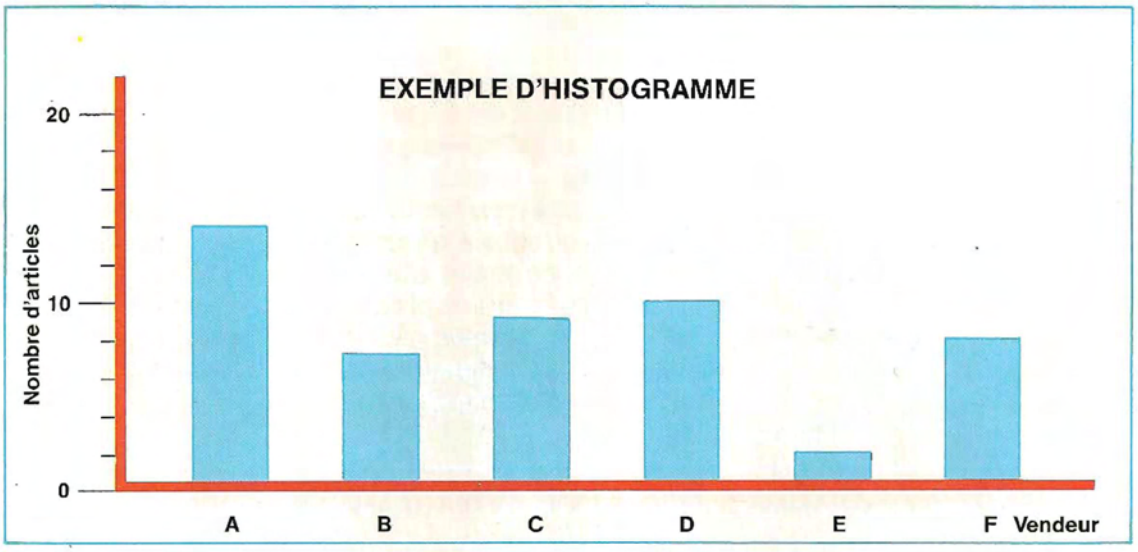
Vendeur	Quantité vendue
A	12
B	7
C	9
D	10
E	2
F	8

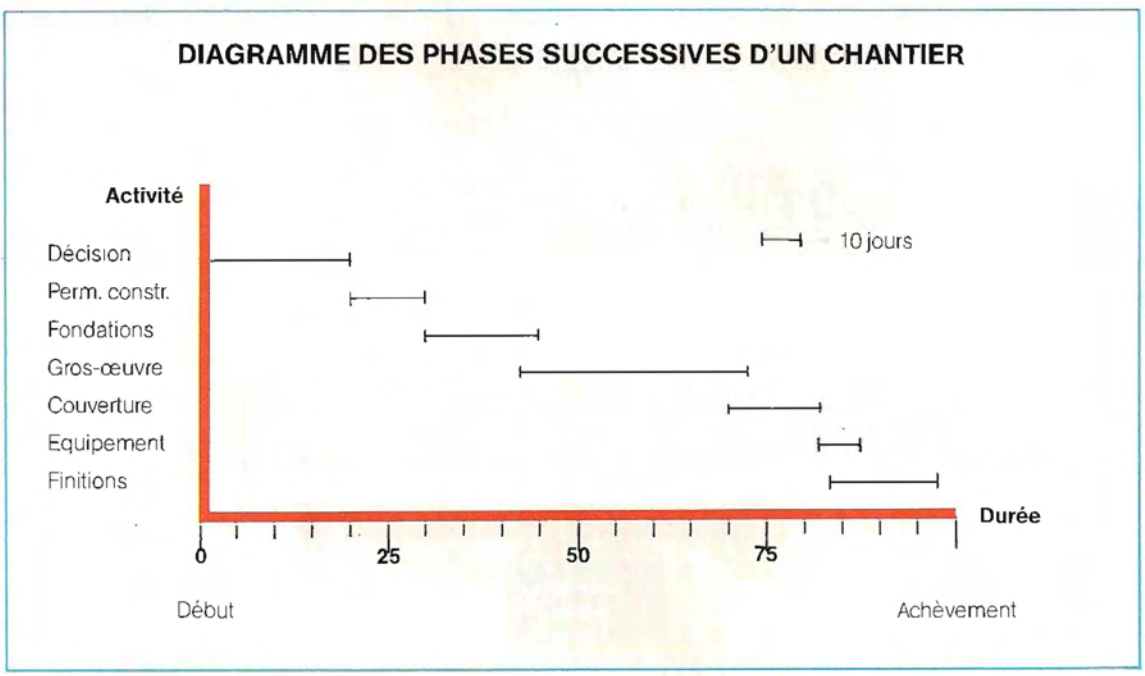
La représentation graphique du haut de la page 589 illustre ce tableau, permettant de mieux visualiser la disparité des résultats et l'hétérogénéité du groupe. On a choisi un histogramme car les vendeurs sont en « quantité discrète » (il n'y a pas d'intermédiaires entre les deux valeurs A et B, par exemple), contrairement aux fonctions que l'on peut représenter par une courbe continue.

Il est évident que dans un cas comme celui-ci, il est impossible de tracer une courbe. Sauf exception, une courbe est en effet définie pour toutes les valeurs comprises entre deux points. Un histogramme ne concerne, au contraire, qu'un certain nombre de valeurs ponctuelles. Nous allons maintenant développer une application concernant un type particulier d'histogramme : **le diagramme à barres.**

Pendant la phase préparatoire d'un projet industriel, on doit rendre compte des temps de réalisation. Le projet le plus simple comporte plusieurs étapes de durée différente,

Imprimante FX 80 Epson.





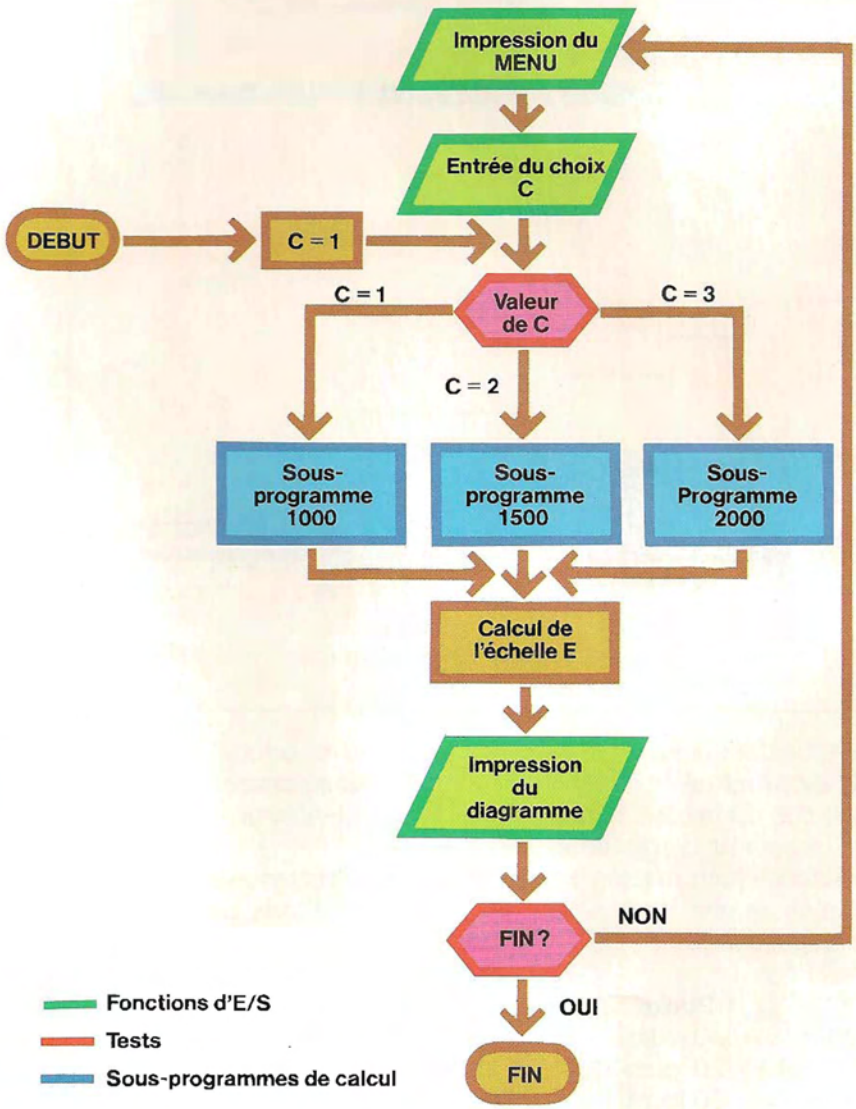
et dont la plupart sont subordonnées aux précédentes. La durée totale de réalisation dépend donc de la succession des différentes étapes, que l'on peut symboliser sur un diagramme. Ainsi, pour la construction d'une maison, on recensera les différentes phases de la réalisation et leur durée respective :

Activité	Durée
Décision	40 jours
Permis de construire	20 jours
Fondations	30 jours
Gros-œuvre	60 jours
Couverture	24 jours
Equipement	16 jours
Finitions	32 jours

En reportant les données de ce tableau sur un diagramme à barres, on obtient le graphique présenté ci-dessus. On remarque que, si certaines phases ne peuvent commencer qu'une fois la précédente terminée, d'autres autorisent un certain chevauchement.
La durée totale de la réalisation n'est donc pas égale au cumul des différentes phases mais à une valeur résultante qui apparaît clairement

sur le schéma de la page 589.
Le recours à de tels diagrammes est assez fréquent dans les programmes d'organisation et de planification disponibles sur les micro-ordinateurs ou les mini-ordinateurs de gestion. Ce type de logiciel permet de prévoir quels seront les besoins en main-d'œuvre à telle ou telle période, ou encore les répercussions sur l'ensemble du projet d'un retard éventuel dans l'une ou l'autre des phases. On peut alors, en simulant diverses causes de retard possibles, évaluer leur retentissement sur la durée des travaux et leurs conséquences économiques.

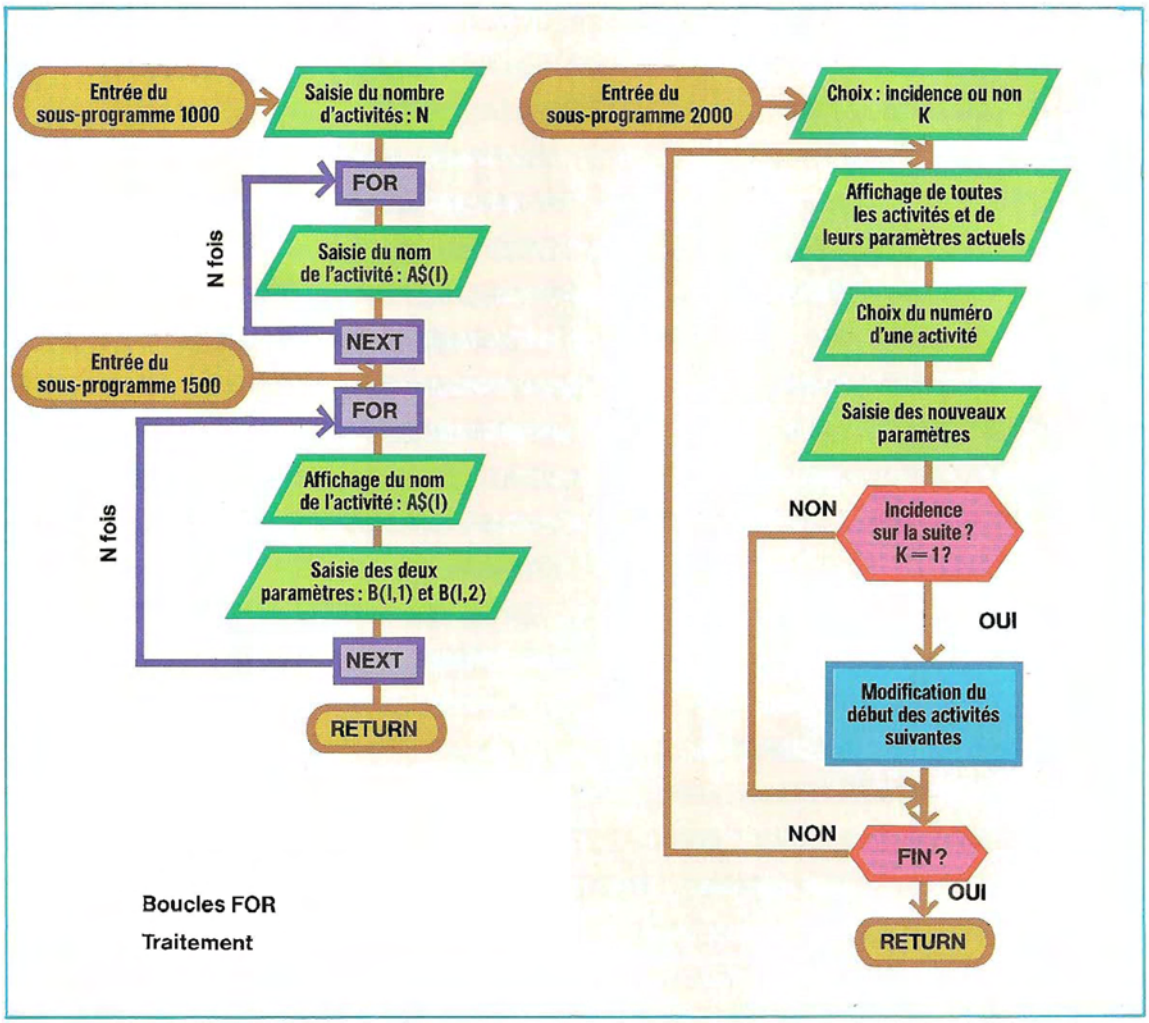
ORGANIGRAMME D'IMPRESSION DE DIAGRAMMES A BARRES



Le MENU est la liste des choix proposés à l'utilisateur. En réponse, on n'a souvent qu'une touche : numéro du choix ou initiale du choix.

Ce test est effectué par l'instruction : ON X GOSUB... qui permet de rajouter des options au menu et d'aiguiller le traitement vers d'autres sous-programmes adaptés.

Le calcul de l'échelle et le tracé du diagramme dépendent de la longueur d'une ligne sur l'imprimante : on a choisi 80 caractères par ligne mais on pourrait très bien augmenter la précision en employant le mode condensé ou une imprimante plus large.



Un programme simple peut permettre d'élaborer des diagrammes à barres et d'obtenir leur tracé par l'intermédiaire de l'imprimante. Voyons comment.

Les diverses activités seront tout d'abord définies puis associées à deux paramètres : leur début et leur durée. Le segment affecté à chacune d'elles sera symbolisé par un certain nombre de caractères unité.

Ce caractère (en l'occurence : *) représentera l'échelle du diagramme en jours, échelle qui sera déterminée de façon à ce que la durée totale du projet tienne dans la largeur de la feuille (quatre-vingts caractères en mode normal d'impression).

De manière à étendre les possibilités de ce programme, on aura recours à un Menu qui proposera à l'utilisateur plusieurs options :

– définition d'une nouvelle série d'activités ;
– conservation des activités précédemment définies, mais modification des paramètres de chacune d'entre elles ;
– modification des paramètres de certaines activités seulement.

Cette dernière possibilité permet de corriger une éventuelle erreur lors de l'entrée des données, mais autorise aussi, si on le désire, une simulation de retard (par exemple). La modification du début ou de la durée d'une des phases est alors répercutée sur toutes les activités ultérieures.

Pour faciliter ces traitements différents selon le choix, les affectations et les calculs sont gérés par des sous-programmes adaptés. Les retours de sous-programme donnent alors lieu à l'impression après calcul de l'échelle E adaptée.

D'autres types de représentations graphiques seront abordés à la fin de l'étude consacrée à la console vidéo.

PROGRAMME DE TRACE DE DIAGRAMMES A BARRES

```
10 ' *** TRACE DE DIAGRAMMES A BARRES
20 '     PROGRAMME : DIAG/BARRES
30 OPTION BASE 1
40 DEFINT A-Z
50 S$="*"
60 I=1 : GOTO 150
70 '
80 ' *** MENU ***
90 PRINT TAB(10);"MENU" : PRINT
100 PRINT " 1 - CHANGEMENT D'ACTIVITES"
110 PRINT " 2 - MODIFICATIONS SUR TOUTES LES ACTIVITES"
120 PRINT " 3 - MODIFICATIONS SUR QUELQUES ACTIVITES"
130 PRINT
140 INPUT "Quel est votre choix ? ";I
150 ON I GOSUB 1000, 1500, 2000
160 '
170 ' CALCUL DE L'ECHELLE ET IMPRESSION
180 E=1
190 IF (B(N,1)+B(N,2) / E > 64 THEN E=E+1 : GOTO 190
200 INPUT "TITRE DU DIAGRAMME : ";T$
210 LPRINT
220 LPRINT T$ : LPRINT
230 FOR I=1 TO N
240 C = ABS(B(I,1)/E)
250 D = ABS(B(I,2)/E)
260 LPRINT A$(I); TAB(15); "!";
270 LPRINT SPACE$(C)
280 LPRINT STRING$(D,S$)
290 NEXT I
300 FOR I=1 TO 80 : LPRINT "-"; : NEXT I
310 LPRINT TAB(16); "Echelle : ";S$;" = ";E
320 LPRINT
330 INPUT "Voulez-vous continuer ? "; REP$
340 IF REP$="OUI" GOTO 90
350 END
1000 '
1010 ' ** SOUS-PROGRAMME 1000 **
1020 INPUT "Nombre d'activités ? "; N
1030 DIM A$(N),B(N,2)
1040 FOR I=1 TO N
1050 PRINT "ACTIVITE "; I;" : (14 caractères maximum)"
1060 INPUT A$(I)
1070 A$(I) = LEFT$(A$,14)
1080 NEXT I
1500 '
1510 ' ENTREE DU SOUS-PROGRAMME 1500
1520 FOR I=1 TO N
1530 PRINT "ACTIVITE "; I; " : "; A$
1540 INPUT "Début des travaux :"; B(I,1)
1550 INPUT "Durée prévue : "; B(I,2)
1560 NEXT I : RETURN
2000 '
2010 ' ** SOUS-PROGRAMME 2000 **
2020 FOR I=1 TO N
2030 PRINT I, A$(I), "DEBUT : "; B(I,1), "DUREE : "; B(I,2)
2040 NEXT I
2050 INPUT "Numéro de l'activité à modifier ? "; J
2060 PRINT A$(J)
2070 INPUT "DEBUT DES TRAVAUX : "; K
2080 INPUT "DUREE PREVUE : "; K
2090 INPUT "Cette modification aura-t-elle une incidence sur la suite ? "; REP$
2100 IF REP$<>"OUI" GOTO 2070
2120 M = K+L-(B(J,1)+B(J,2)
2130 FOR I=J TO N
2140 B(I,1) = B(I,1)+M
2150 NEXT I
2160 B(J,1) = K : B(J,2) = L
2170 INPUT "Désirez-vous modifier une autre activité ? "; REP$
2180 IF REP$="OUI" GOTO 2020
2200 RETURN
```

```
DIAGRAMME A BARRES : ESSAI 1

PROJET          !  **********
REVISIONS       !           *****
CALCULS         !             ********
TERRASSEMENT    !                     *****
FONDATIONS      !                          **********
MURS            !                                *****
TOITURE         !                                     **********

          Echelle : * = 2 jours

DIAGRAMME A BARRES : RETARD  EN DEBUT DE CHANTIER

PROJET          !  **********
REVISIONS       !           *****
CALCULS         !             ********
TERRASSEMENT    !                     ********
FONDATIONS      !                           **********
MURS            !                                *****
TOITURE         !                                     **********

          Echelle : * = 2 jours
```

Test 17

1 / Une imprimante présente les caractéristiques suivantes :
- tête d'impression à aiguilles
- bi-directionnelle
- 132 colonnes
- 120 cps
- buffer 4K

Que signifient ces indications ?

---

2 / Quels caractères seront imprimés si on transmet à l'imprimante les commandes suivantes ?

CHR$(33) CHR$(110) CHR$(27) CHR$(10)

---

3 / Qu'imprime le programme suivant ?

```
10 A$="ESSAI" :B=1270 :C=10
20 PRINT A$,C
30 PRINT USING "####.#";A$;B
```

---

4 / On demande l'édition de la table A(5) à l'aide des instructions suivantes :

```
10 FOR I=1 TO 5 : LPRINT A(I) : NEXT I
20 FOR I=1 TO 5 : LPRINT A(I) ; : NEXT I
30 FOR I=1 TO 5 : LPRINT TAB(10) ; A(I) ; : NEXT I
```

Que va-t-on obtenir pour chacune d'elles ?

Les solutions du test se trouvent page 599.

Gestion de la console vidéo

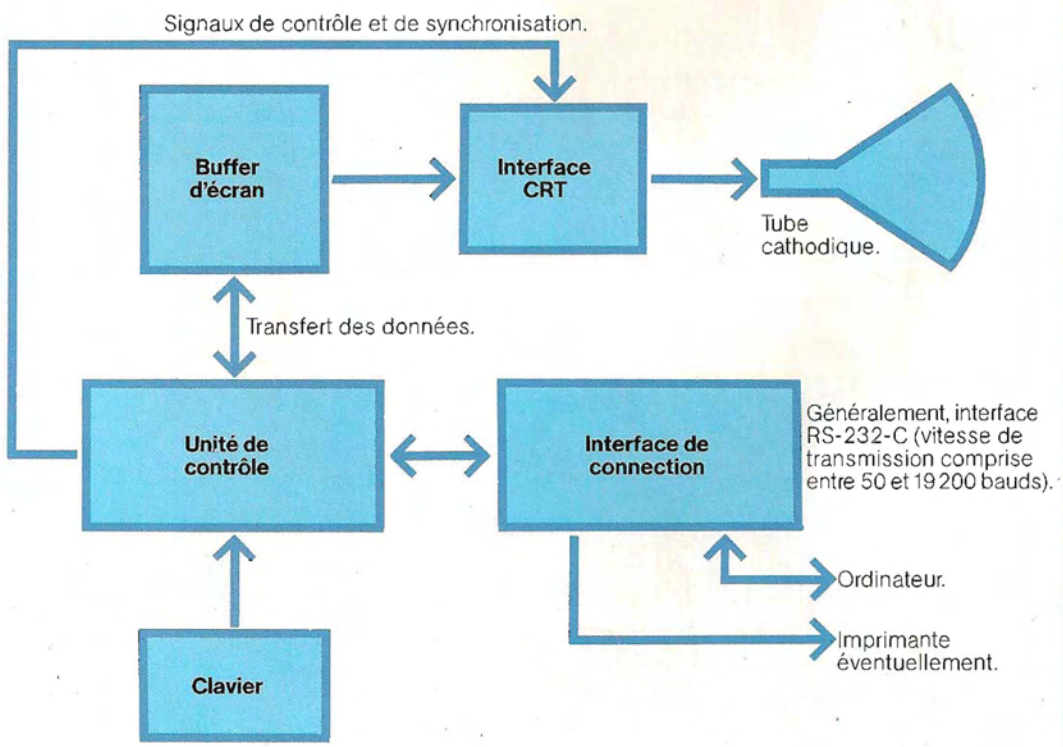
L'évolution des micro-ordinateurs a conduit à la configuration de base actuelle qui adopte la console vidéo (clavier et écran) comme principal organe d'entrée/sortie.
La commodité et la fonctionnalité de ce périphérique sont aujourd'hui universellement reconnues et son domaine d'utilisation s'est même étendu aux terminaux des gros systèmes.
Bien connaître les possibilités de la console vidéo permet d'en tirer meilleur parti et notamment de travailler en mode conversationnel, établissant ainsi un véritable dialogue avec l'unité centrale.
Toutefois, c'est surtout au programmeur qu'il incombe de déterminer le mode de gestion qui exploitera au mieux les capacités du matériel.
En effet, lors des applications spécifiques comme, entre autres, la saisie ou la présentation de données, une bonne maîtrise de la console vidéo constitue toujours un avantage appréciable.

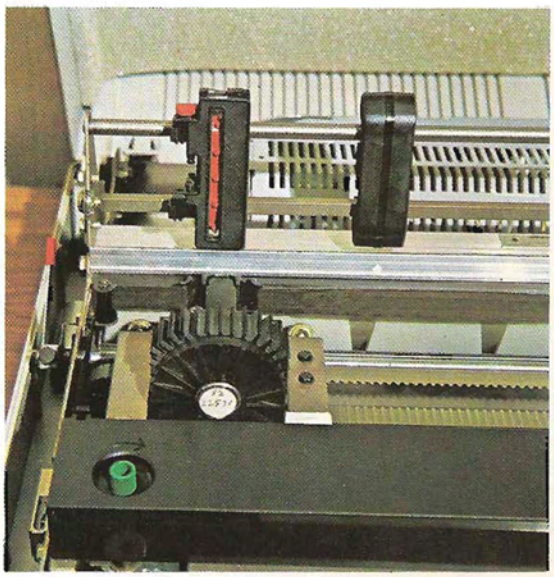
Structure et caractéristiques d'une console vidéo

Le principe de fonctionnement d'une telle console est schématisé au bas de cette page. Nous allons reprendre un à un les principaux éléments de cet ensemble pour décrire brièvement leur utilité.

- **Le clavier** est l'organe d'entrée. A la frappe d'une touche, le code ASCII correspondant est généré et transmis à l'unité de contrôle ;
- **L'unité de contrôle** a un rôle de coordination et de synchronisation des autres organes. Elle assure l'interprétation et l'exécution des commandes, et contrôle les communications avec l'ordinateur ;
- **L'interface de connection** gère le transfert des données de l'ordinateur au périphérique et réciproquement. Certaines consoles sophistiquées disposent d'une interface permettant de les connecter directement à une imprimante. Ce système décharge l'unité centrale et autorise une transcription fidèle du texte affiché : la copie d'écran.

PRINCIPE DE FONCTIONNEMENT D'UNE CONSOLE VIDEO





V. Pirozzi/Archives Curcio

Sous le couvercle d'une imprimante.

- **Le buffer d'écran** est un bloc de mémoire RAM où est stockée, sous forme codée, l'image de ce qui doit apparaître sur l'écran. La capacité de cette mémoire tampon dépend du nombre de caractères affichables. Ainsi, pour un écran de 80 colonnes × 24 lignes, qui peut donc contenir 1920 caractères, le buffer doit correspondre à 1920 emplacements.

C'est volontairement qu'on a employé le terme d'emplacement et non celui d'octet, car, dans ce cas précis, un emplacement n'occupe pas forcément un octet. En effet, pour chaque caractère à afficher, on doit mémoriser non seulement les 8 bits du code ASCII mais éventuellement aussi des informations complémentaires ou **attributs** modifiant les caractéristiques de présentation (vidéo normale ou inverse, clignotement, soulignement, etc.). Cela peut nécessiter jusqu'à 4 bits supplémentaires.
La longueur d'un emplacement de mémoire varie donc d'un écran à l'autre selon les possibilités offertes.
Ces indications ont une valeur générale, mais certains matériels adoptent des caractéristiques très différentes, notamment lors d'utilisation orientée vers le mode graphique.
Ces écrans à vocation particulière feront ultérieurement l'objet d'une présentation spécifique.

- **L'interface CRT** (Cathode Ray Tube) transforme le code de chaque emplacement par l'intermédiaire d'un générateur de caractères. Les signaux vidéo ainsi obtenus sont alors dirigés vers le tube cathodique.

La console autorise plusieurs modes de fonctionnement adaptés aux besoins des différentes applications et basés sur le protocole de transmission des commandes ou données de la console vers l'unité centrale.

CONVERSATIONNEL. Les commandes sont envoyées directement à l'unité centrale dès l'entrée au clavier. Il s'agit le plus souvent d'une transmission asynchrone et en série.
Ce mode comporte deux variantes :

le semi-duplex : liaison bi-directionnelle non simultanée où le terminal exécute les commandes provenant soit de l'unité centrale, soit du clavier ;

le duplex intégral : liaison bi-directionnelle simultanée où les commandes entrées au clavier sont toutes transmises à l'ordinateur avant d'être exécutées. La console est donc sous le contrôle intégral de l'ordinateur.

PAGE. Ce mode très particulier permet de transmettre les données à l'ordinateur non plus une à une ou ligne par ligne, mais page par page. Une page correspond à la totalité de l'écran. Ce mode n'est généralement pas accessible à partir du Basic, ni des autres langages évolués. Il peut cependant être mis en place par des routines spéciales rédigées en langage Assembleur.

SCROLL. Ce mode est très proche du mode PAGE mais n'est pas limité au contenu de l'écran. Alors que, dans le mode précédent, le curseur se bloque en bas et à droite après la frappe du dernier caractère, le SCROLL permet de continuer en faisant défiler les lignes. La première disparaît et tout le texte est décalé vers le haut, libérant une nouvelle ligne en bas de l'écran.

PROTECT. Ce mode permet d'assurer la protection de caractères ou groupes de caractères en leur accordant un attribut particulier. Il sera alors impossible à l'opérateur de modifier ces caractères car le curseur ne s'arrête pas sur eux.

Ces différents modes de fonctionnement sont commutables, à partir du programme ou en direct, par l'emploi de commandes spécifiques. Comme celles destinées à l'imprimante, ces commandes sont normalement introduites par un caractère de contrôle: ESCape, de code ASCII 27. Ce caractère informe le terminal que les caractères qui le suivent doivent être interprétés comme des commandes et non comme des données.

Gestion du curseur

Des terminaux vidéo très sophistiqués vont jusqu'à permettre des modifications de la forme du curseur. Les commandes que nous allons citer en exemple sont celles du terminal Seletron 3410. Bien que leur gamme soit particulièrement fournie, leur mise en œuvre est généralement représentative des solutions les plus classiques. Voici cependant quelques cas particuliers. Par exemple, un curseur rectangulaire constitué d'une matrice 9 × 11. On peut tout d'abord intervenir sur le mode d'affichage de ce curseur et opter pour :

- l'affichage fixe ;
- l'effacement du curseur ;
- le clignotement (un réglage de la fréquence est possible sur certains matériels).

Pour modifier la forme du curseur, on précisera simplement lesquelles des neuf lignes et des onze colonnes de la matrice doivent être visualisées (toutes à l'origine). Les commandes sont transmises sous la forme habituelle :

ESC + "paramètres"

Ainsi, sur cette machine, "c0" est le code de l'affichage fixe et "c3" le code du clignotement rapide. L'instruction prend alors la forme :

```
PRINT CHR$(27)+"c0"
```

Le curseur sera donc fixe et le restera jusqu'à l'envoi d'une commande contraire telle que :

```
PRINT CHR$(27)+"c3"
```

Cette dernière déclenche le clignotement rapide du curseur qui reste actif jusqu'à la modification suivante. .

Un peu plus complexe est l'instruction produisant une modification de la forme du curseur. Elle se présente ainsi :

```
PRINT CHR$(27)+"c4SE"
```

S est le numéro de la première ligne de la matrice du curseur devant être visualisée. E est le numéro équivalent pour la première colonne.

Page 597 se trouve le listing d'un programme de démonstration des diverses possibilités d'affichage. Comme nous l'avons vu, celles-ci sont sélectionnées par des commandes qui sont explicitées dans les lignes 44 à 58 du listing. La ligne 100 définit une fonction qui associe le caractère ESCape et une chaîne F$(N) elle-même définie grâce aux différents codes commande mémorisés en DATA (lignes 140 à 210).

Pour obtenir le mode d'affichage désiré, on donne à N la valeur voulue et on demande l'impression de FNC$(N). Si, par exemple, on affecte 3 à N, on prélèvera la chaîne F$(3), c'est-à-dire "G1", qui est le code du soulignement.

Voici donc les instructions qu'il convient d'utiliser pour qu'un texte apparaisse souligné à l'écran :

```
N = 3
PRINT FNC$(N);"Essai de soulignement"
```

Le programme présente également une fonction de positionnement du curseur qui permet d'écrire le texte à une position précise de l'écran. Soit les instructions suivantes :

```
10  A$="Essai"
20  X=5 :Y=10
30  PRINT FNP$(X,Y);A$
```

Elles déterminent l'affichage de la chaîne "Essai" à partir de la 5e colonne (X=5) de la 10e ligne (Y=10).

Les masques de saisie

Ces nombreuses possibilités de contrôle de l'affichage sont à l'origine des « masques de saisie ».

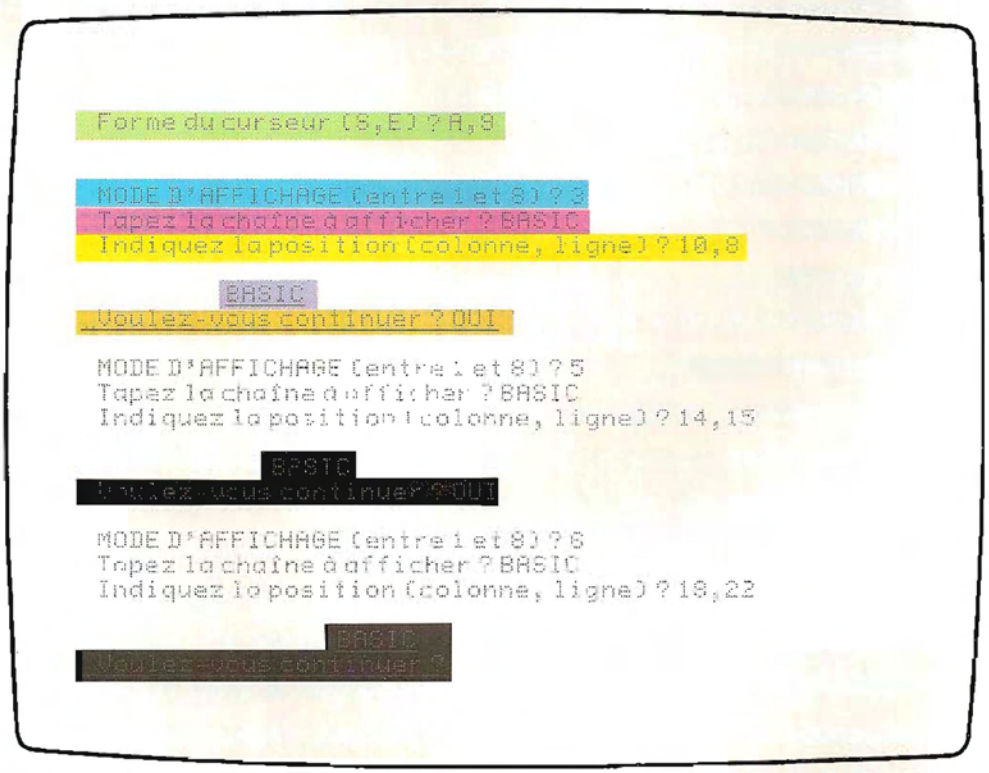
Ces masques confèrent une grande souplesse aux programmes d'acquisition de

EXEMPLES DE GESTION DE L'AFFICHAGE

```
10 ' *** Exemples de gestion de l'affichage
20 '     PROGRAMME : AFFICH.DEMO
25 OPTION BASE 1
30 DEFINT A-Z
42 '    COMMANDES                          EFFET
44 '    CHR$(27)+")"          Affichage en demi-intensité
46 '    CHR$(27)+"("          Retour à la luminosité normale
48 '    CHR$(27)+"G1"         Soulignement
50 '    CHR$(27)+"G2"         Clignotement
52 '    CHR$(27)+"G4"         Vidéo inverse (caractères noirs sur blanc)
54 '    CHR$(27)+"G5"         Soulignement et vidéo inverse
56 '    CHR$(27)+"G8"         Clignotement et vidéo inverse
58 '    CHR$(27)+"G0"         Retour à la normale
60 '
70 ' * FORME DU CURSEUR : CHR$(27)+"c4"+"S"+"E"
80 '
90 '
100 DEF FNC$(N)=CHR$(27)+F$(N)    ' Mode d'affichage (lignes 44-58)
110 DEF FNF$(S$,E$)=CHR$(27)+"c4"+S$+E$    ' Forme (ligne 70)
120 ' VALEURS DES CHAINES F$ POUR LESQUELLES LA FONCTION EST DEFINIE
125 RESTORE 140
130 FOR I=1 TO 8 : READ F$(I) : NEXT I
140 DATA")"
150 DATA"("
160 DATA"G1"
170 DATA"G2"
180 DATA"G4"
190 DATA"G5"
200 DATA"G6"
210 DATA"G0"
220 '
230 ' *** POSITIONNEMENT DU CURSEUR : X=COLONNE Y=LIGNE
240 DEF FNP$(X,Y)=CHR$(27)+CHR$(61)+CHR$(31+Y)+CHR$(31+X)
250 '
260 ' ** FONCTIONS SPECIALES
270 '
280 BLK$=CHR$(27)+"+"          ' Effacement de l'écran
290 BEL$=CHR$(7)               ' Emission d'un bip
300 '
310 ' ** CHOIX DE LA FORME DU CURSEUR ET DE SON POSITIONNEMENT
320 '
325 X=1:Y=1: PRINT FNP$(X,Y);
330 INPUT "Forme du curseur (S,E) ";S$,E$
360 PRINT FNF$(S$,E$)          ' On donne au curseur la forme voulue
370 '
380 ' * MODE D'AFFICHAGE
390 PRINT FNC$(8)          ' RETOUR A LA NORMALE
400 INPUT " MODE D'AFFICHAGE (entre 1 et 8) ";N    ' Voir les DATA
410 IF N>=1 AND N<=8 THEN GOTO 500    ' Valeur correcte
420 ' ** ERREUR **
430 '
440 X=35:Y=12 PRINT FNP$(X,Y)    ' Curseur au centre de l'écran
450 N=6: PRINT FNC$(N);" ** ERREUR **"
460 FOR I=1 TO 800: NEXT I          ' Boucle de "temporisation"
461 S$=INPUT$(1)
470 GOTO 260
480 '
500 ' ** ENTREE D'UN TEXTE **
510 '
520 INPUT " Tapez la chaîne à afficher ";A$
530 INPUT " Indiquez la position (colonne, ligne) ";X,Y
540 K=LEN(A$)          ' Longueur de la chaîne à afficher
550 M=80-K             ' On vérifie que la chaîne peut tenir sur la ligne
560 IF X<0 OR X>M THEN PRINT " ** ERREUR **": GOTO 520
570 IF Y<0 OR Y>24 THEN PRINT " ** ERREUR **": GOTO 530
580 '
590 ' **** AFFICHAGE ****
610 PRINT FNP$(X,Y); FNC$(N);A$
640 INPUT " Voulez-vous continuer ? ";REP$
650 IF REP$="OUI" THEN GOTO 390
660 END
```

EXEMPLE DE GESTION D'ECRAN

Ces illustrations montrent quelques modes d'affichage obtenus grâce au programme de la page précédente.



A l'exécution de la ligne 330, les paramètres qui définissent la forme du curseur sont demandés à l'opérateur. Celui-ci a tapé les chiffres hexadécimaux A et 9 (10x9), qui donnent au curseur une forme rectangulaire.

A la ligne 400, le programme demande l'entrée du code du mode d'affichage. La valeur entrée (3) correspond au soulignement (ligne 48).

L'instruction 520 demande la frappe de la chaîne à afficher. Ici, la chaîne « BASIC » est affectée à la variable A$.

Le programme demande maintenant à l'opérateur les coordonnées du point de l'écran où devra commencer la chaîne A$ (ligne 530). L'opérateur choisit l'intersection de la colonne 10 et de la ligne 8.

L'instruction PRINT de la ligne 610 est exécutée. Le mot BASIC, entré auparavant par l'opérateur, apparaît souligné sur l'écran.

A la demande du programme (ligne 640), l'opérateur exprime son désir de continuer. La ligne 650 provoque le branchement à la ligne 390.
La suite montre deux autres modalités d'affichage de la même chaîne : d'abord, en vidéo inverse, puis toujours en vidéo inverse mais avec soulignement.