Susan Curran Ray Curnow

PREMIERS PAS EN BASIC

HACHETTE
Informatique

ACCES

Susan Curran Ray Curnow

PREMIERS PAS EN BASIC

Traduit par Hervé Devaine

HACHETTE
Informatique

Traduction de l'ouvrage publié en langue anglaise
sous le titre
FIRST STEPS IN BASIC
dans la collection :
The Clear and Simple Home Computer Series
par Frances Lincoln Ltd,
Apollo Works, 5 Charlton King's Road, London NW5 2 SB,
pour le compte de Windward, W H Smith & Son, Ltd.

Table des matières

Introduction

Cet ouvrage a pour but de vous enseigner le Basic, le langage de presque tous les ordinateurs familiaux. Nous vous apprendrons à écrire et à adapter des programmes simples pour votre machine.

Quel ordinateur utilisons-nous ?

Nous ferons référence à deux ordinateurs : le nôtre et le vôtre. Si vous n'en possédez pas encore ou si vous ne pouvez en utiliser qu'occasionnellement, à l'école ou à votre travail, ne vous inquiétez pas ; l'étude de ce livre vous sera facilitée par les applications effectuées sur notre ordinateur. Toutefois, la programmation est un art difficile sans machine. Tant qu'on ne l'a pas exécuté, on ne peut être assuré du bon fonctionnement d'un programme. En conséquence, si vous ne disposez pas d'une machine, nous vous suggérons de lire l'ouvrage dans sa totalité et d'en pratiquer les exercices. Vous vous familiariserez ainsi avec le Basic et c'est en connaissance de cause que vous prendrez la décision d'acquérir un micro-ordinateur.

Lisez l'annexe 1, les indications que vous y trouverez, jointes à la lecture de magazines et d'ouvrages consacrés à la micro-informatique, complétées par les conseils d'un revendeur, vous aideront dans votre choix.

Bien que les ordinateurs familiaux soient dans l'ensemble similaires, ils se distinguent les uns des autres par toute une série de caractéristiques, et, notamment, par leur Basic. La plupart des ouvrages consacrés à ce langage font référence à un ordinateur bien précis, ce qui rend difficile l'application des connaissances acquises à une autre machine. C'est pourquoi le micro-ordinateur que nous utilisons est totalement hybride, c'est-à-dire, qu'empruntant ses caractéristiques à plusieurs machines familiales, il en est la résultante. Ainsi, nous avons extrait les points communs à plusieurs Basics pour élaborer la version qui vous est présentée ici. Elle offre les avantages suivants :

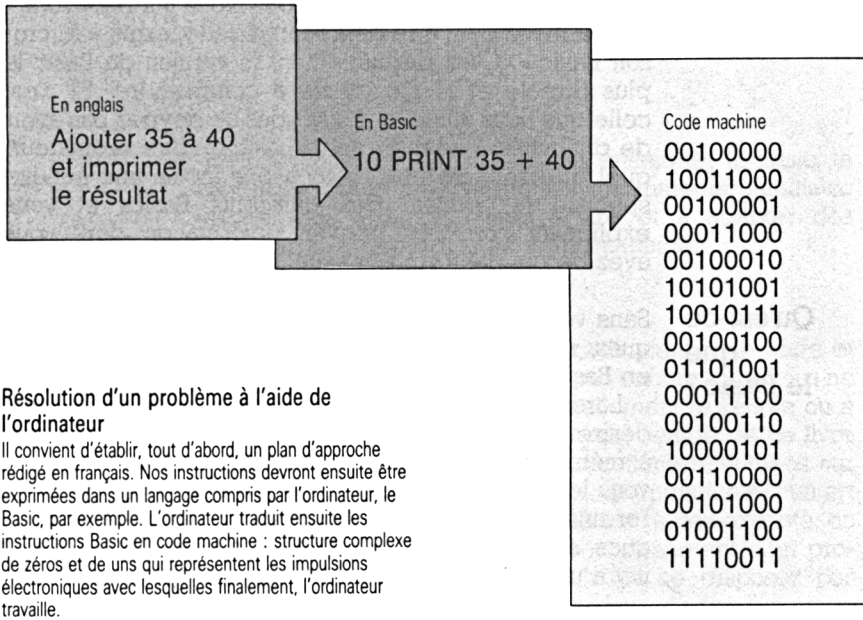
— Les commandes utilisées se retrouvent sur la plupart des ordinateurs familiaux.
— Elle peut être facilement adaptée à votre machine.
— En l'utilisant, vous apprendrez à écrire des programmes qui, nous le pensons, vous intéresseront.

Vous avez peut être déjà rencontré le terme « Microsoft Basic ». C'est probablement la version du Basic la plus directe et la plus facile à comprendre. Et c'est celle que nous utilisons. Ainsi, vous ne devriez pas avoir de difficultés à adapter notre Basic à votre ordinateur, quel qu'il soit. De plus, nous avons inclus une annexe spéciale (1) traitant des différents Basics et vous expliquant comment trouver l'information dont vous avez besoin dans votre manuel d'utilisation.

Qu'est-ce que le Basic ?

Sans vouloir vous rebuter avec trop de détails techniques, nous allons vous expliquer ce que programmer en Basic veut dire.

Lorsque vous réfléchissez à une tâche que vous désirez confier à votre ordinateur — un calcul, une liste à mémoriser, un dessin à effectuer sur écran, etc. — vous le faites en français. Il n'en va pas de même pour l'ordinateur. Il fonctionne avec des impulsions électroniques, sur un mode dit « binaire », à deux états. Quand il y a une impulsion électrique, il s'agit du premier état — 1 —, quand il n'y a pas d'impulsion, c'est l'état 0. On a imaginé un grand nombre de systèmes ingénieux fondés sur des assemblages de 0 et de 1. L'ordinateur en les combinant est ainsi en mesure d'effectuer toutes les tâches qui lui sont confiées. Par bonheur, pour les lui faire accomplir, nous ne sommes pas obligés d'employer ces structures à base de 0 et de 1. En effet, à l'intérieur des ordinateurs qui utilisent des langages dits « évolués », tel le Basic, se trouvent des circuits électriques qui leur permettent de comprendre et de décomposer les instructions données en Basic en une suite de 0 et de 1. Les langages pour ordinateur, contrairement au français, ont un vocabulaire très limité. Si vous désirez résoudre un problème, celui-ci doit être fractionné en une série d'instructions qui puissent être traduites à leur tour dans le langage de la machine. Les instructions, qui sont ensuite entrées dans l'ordinateur, forment un programme. Ainsi apprendre la programmation, c'est apprendre à décomposer un problème en instructions que l'ordinateur peut manipuler, à transposer celles-ci dans le langage machine, et bien sûr, à les tester.



Résolution d'un problème à l'aide de l'ordinateur

Il convient d'établir, tout d'abord, un plan d'approche rédigé en français. Nos instructions devront ensuite être exprimées dans un langage compris par l'ordinateur, le Basic, par exemple. L'ordinateur traduit ensuite les instructions Basic en code machine : structure complexe de zéros et de uns qui représentent les impulsions électroniques avec lesquelles finalement, l'ordinateur travaille.

Les concepteurs de langages ont donné au Basic ainsi qu'à différents langages utilisés par d'autres ordinateurs, une forme qui s'apparente le plus possible à la langue anglaise. Mais ce n'est pas tout à fait de l'anglais. L'ordinateur a l'esprit littéral et tout ce que l'on écrit en Basic doit être traduit en structures de 1 et de 0. Par conséquent, il faut que les instructions données à l'ordinateur soient claires, sans ambiguïté aucune, et qu'elles respectent les règles de la syntaxe. De prime abord, ces règles (utilisation de la virgule, du point-virgule, etc.) vous sembleront un peu complexes. Mais petit à petit, leur emploi vous deviendra plus naturel. Plus vous vous exercerez, plus il vous sera facile d'exprimer vos idées en Basic. Le schéma de la page 10 vous montre sommairement la manière de penser un problème et de le faire résoudre et exécuter par l'ordinateur.

L'ordinateur comprend-il ce que vous lui dites ?

Si vous ne respectez pas les règles du Basic, votre ordinateur vous le fera savoir rapidement. Voici quelques indications qui vous permettront de détecter les anomalies.

1) Si votre ordinateur fait ce que vous espérez (ou, quand nous débuterons, ce que nous vous disons d'espérer) et, soit ne vous donne aucun message, soit affiche quelque chose comme OK ou READY, alors c'est que tout va bien.

2) Si votre ordinateur :

— fait ce que vous escomptiez, mais vous donne un message d'erreur (dans le cas où vous éprouveriez des difficultés à l'interpréter, consultez le chapitre 5) ;

— ne fait rien du tout (à moins que ce ne soit là ce que vous vouliez) ;

— fait quelque chose de différent de ce à quoi vous vous attendiez, c'est que vous avez commis une erreur dans votre instruction ou dans votre programme. Essayez de retaper l'instruction : il s'agit peut-être d'une erreur de frappe. Si malgré cela votre ordinateur ne l'accepte pas, vérifiez tout le programme.

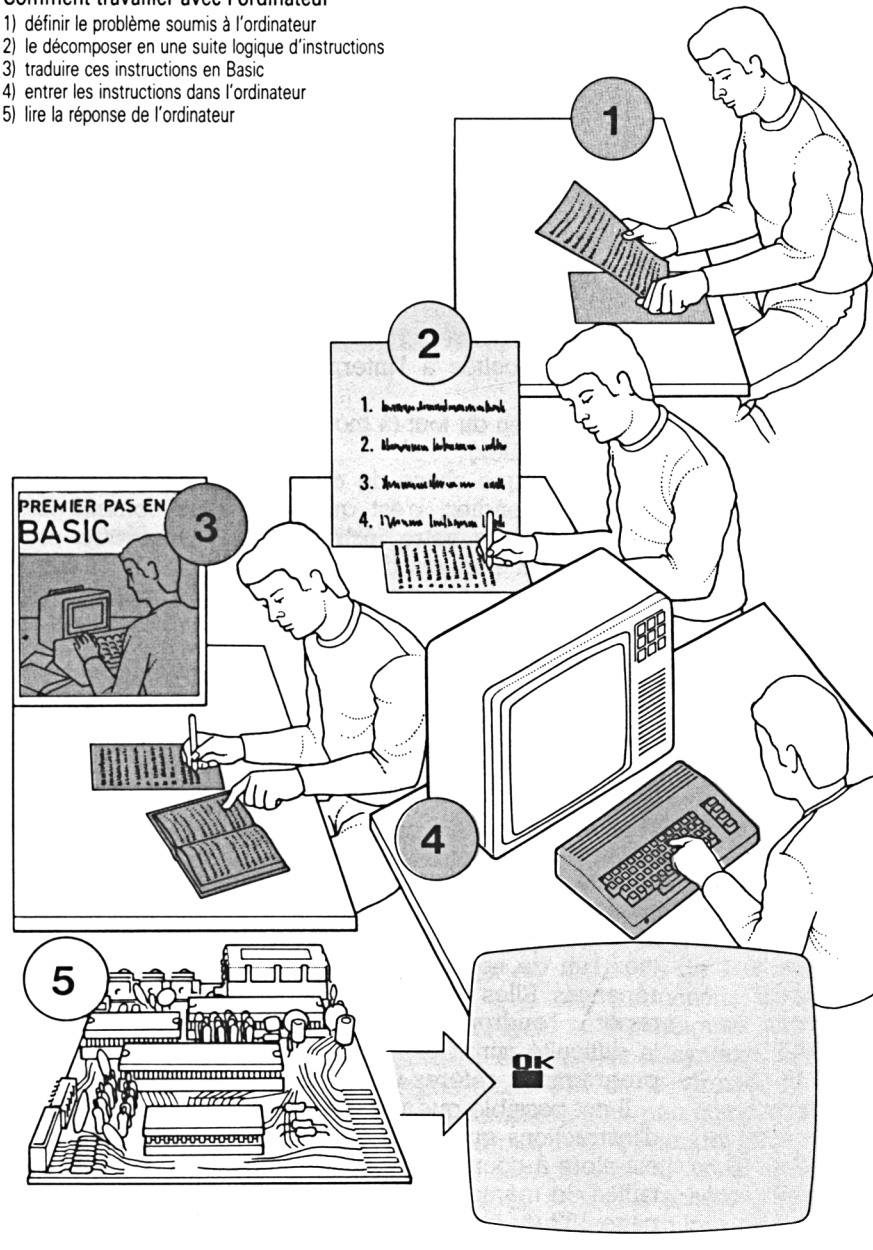
Et dites-vous bien que ce genre d'incidents n'épargne même pas les programmeurs les plus chevronnés. La source de vos difficultés peut également résider dans les différences qui séparent votre Basic du nôtre. L'annexe 1 devrait vous aider à corriger ces divergences.

Il se peut également qu'une petite erreur se soit glissée dans l'élaboration de votre programme ou dans sa rédaction. Si cela vous arrive alors que vous effectuez l'un de nos exercices, jetez un coup d'œil à nos réponses. Elles vous fourniront peut-être un indice pour repérer l'endroit où vous avez commis votre erreur. Si la difficulté surgit alors que vous écrivez vos propres programmes, référez-vous aux conseils du chapitre 5.

Il est possible que vous ayez atteint le nombre limite d'instructions que votre ordinateur peut mémoriser : il est alors à court d'espace mémoire. Nous parlons des tailles de mémoire et des programmes plus longs à la page 152 du chapitre 7, mais vous ne devriez pas être confrontés à ce genre de problème dans les premiers chapitres.

Comment travailler avec l'ordinateur

1) définir le problème soumis à l'ordinateur
2) le décomposer en une suite logique d'instructions
3) traduire ces instructions en Basic
4) entrer les instructions dans l'ordinateur
5) lire la réponse de l'ordinateur



Nous savons combien il peut être frustrant, à chaque programme que l'on écrit, de découvrir qu'il ne fonctionne pas comme il le devrait. Nous avons conçu ce livre de telle façon que cela ne se produise pas trop souvent au cours de votre étude. Si ce devait pourtant être le cas, dites-vous au moins qu'ainsi, vous apprenez les messages d'erreur !

Il ne fait pour nous aucun doute, qu'à l'issue de cet ouvrage, vous serez capable d'écrire toutes sortes de programmes : jeux, calculs, stockage et classement d'informations. Vous n'aurez alors qu'un seul désir : approfondir vos connaissances en Basic pour écrire des programmes plus ambitieux. Afin de vous y aider, le dernier chapitre contient une sélection de programmes. Ce ne sont pas de simples exercices, ce sont de vrais programmes qui vous montrent qu'avec les seules connaissances acquises dans ce livre, vous serez en mesure de réaliser beaucoup de choses avec votre micro-ordinateur. Certains d'entre eux sont abondamment annotés. Vous y trouverez des astuces de programmation dont vous tirerez le plus grand profit.

Terminons-en avec l'introduction. Nous allons maintenant entrer dans le vif du sujet et faire de vous un programmeur Basic.

1

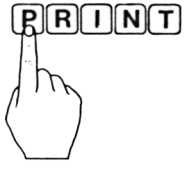
Écrire sur l'écran

Au cours de ce chapitre, nous apprendrons :
— à utiliser les instructions PRINT, PRINT @ ;
— à utiliser la commande CLS,
— à effectuer des calculs simples à l'aide de l'ordinateur,
— à afficher des messages sur l'écran,
— à disposer les sorties sur l'écran.

Si vous avez l'intention de vous servir de votre propre ordinateur pour appliquer les exemples contenus dans cet ouvrage, voici le moment venu. Mettez-le en marche, en respectant bien le mode d'emploi. Sur certains ordinateurs vous devez charger les programmes qui leur permettent de comprendre le Basic. Ce chargement s'effectue à l'aide de disquettes ou de cassettes ; il faut à nouveau suivre les instructions du manuel. Un crayon et du papier seront peut-être les bienvenus pour noter ce que vous découvrirez.

Au risque de nous répéter : ne vous inquiétez pas si votre ordinateur ne réagit pas comme le nôtre. Si vous ne comprenez pas bien ce qu'il fait, ni pourquoi il le fait, alors relisez les instructions ou consultez l'annexe 1. Gardez le mode d'emploi de votre ordinateur par devers vous, cela vous aidera beaucoup.

Maintenant, au travail. Trouvez la touche du clavier (sur votre machine ou sur votre modèle de clavier page 183) qui comporte l'indication RETURN, ENTER ou NEWLINE. C'est une touche très importante. La nôtre est ENTER, nous vous indiquerons quand l'utiliser par ce symbole : {E}



Essayez :

```
PRINT 1 {E}
```

Chaque fois que vous verrez « Essayez », il vous faudra taper ce qui suit sur votre clavier. Si vous n'avez pas d'ordinateur, nous vous indiquerons ce qui se passe sur le nôtre.

« PRINT 1 » est un exemple d'instruction en Basic. « PRINT » est un mot clé qui donne l'ordre à l'ordinateur d'inscrire quelque chose à l'écran ; « 1 » s'appelle une expression : dans le cas présent, c'est ce que la machine va afficher.

Voici ce que l'on obtient sur l'écran :

```
1
OK
```

« OK » est le message que votre ordinateur communique lorsqu'il a terminé ce que vous lui avez demandé de faire. D'autres ordinateurs vous répondront READY. Cela vous avertit qu'il ne fera plus rien jusqu'à ce que vous lui donniez de nouvelles instructions. C'est donc à votre tour de faire quelque chose.

Les instructions Basic débutent par un mot clé, qui peut être suivi d'une expression, ou de quelque chose d'autre (nous y viendrons bientôt), ou de rien du tout.

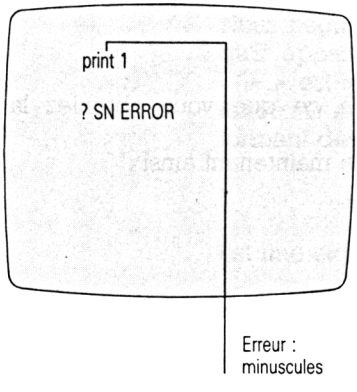
Voici un autre exemple :

```
print 1 {E}
```

(Si votre ordinateur n'a pas de minuscules, alors passez cet exemple.)

Notre ordinateur refuse les instructions tapées en minuscules. Il réagit en affichant un message d'erreur :

```
? SN ERROR
```



Erreurs

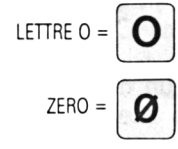
Si l'ordinateur n'accepte pas l'instruction qui lui est donnée, il affiche un message d'erreur. Ici, notre machine refuse les minuscules ; elle nous fait donc savoir que nous nous sommes trompé. Si vous commettez des erreurs, ne vous inquiétez pas, vous ne pouvez pas endommager votre ordinateur en lui fournissant de mauvaises instructions. Vérifiez vos instructions et essayez à nouveau...

Aussi, à partir de maintenant, nous utiliserons les majuscules. Certains ordinateurs acceptent les instructions tapées en minuscules. Si c'est le cas du vôtre, il vous sera alors loisible d'utiliser les minuscules là où nous utiliserons les majuscules.

Si votre ordinateur n'a accepté ni l'une ni l'autre de ces instructions, alors c'est qu'il ne comprend pas le Basic ! Reportez-vous à votre mode d'emploi ou rendez-vous chez votre revendeur pour obtenir des éclaircissements. Vous ne pouvez pas utiliser votre ordinateur s'il ne comprend pas l'une de ces deux instructions. Bien sûr, l'ordinateur peut imprimer bien d'autres choses encore :

Essayez :

```
PRINT 1234567890 {E}
```



Vous venez de découvrir à quoi ressemblent les différents chiffres sur l'écran. Ils se sont probablement inscrits sous une forme peu familière, semblable à celle-ci :

```
1.23456789E + 09
```

Ceci signifie que le nombre est trop grand pour que l'ordinateur puisse le traiter normalement. Il a donc utilisé une notation scientifique. Si vous ne comprenez pas, ne vous inquiétez pas, ce n'est pas d'une importance cruciale. Pour connaître la longueur maximale des nombres acceptés par votre ordinateur,

Essayez :

```
PRINT 12345
PRINT 123456
```

et ainsi de suite, jusqu'à ce que vous obteniez la notation scientifique.

Notre écran se présente maintenant ainsi :

```
PRINT 1
 1
OK
print 1
? SN ERROR
PRINT 1234567890
```

```
 1.23456789E + 09
OK
PRINT 123456789
 123456789
OK
```

Remarquez que le « l » est situé sous le « R » de PRINT et non pas sous le « P »; pourquoi? Notre ordinateur aime donner à tous les nombres un signe (+ ou −). En fait, il n'imprime pas le signe +, mais il laisse un espace libre à la place de ce dernier, et décale le nombre d'un espace vers la droite.

Essayez :

```
PRINT — 12345  {E}
```

Voici ce que nous obtenons sur l'écran :

```
PRINT - 12345
- 12345
OK
```

Le signe − est placé à l'endroit où se trouvait l'espace vide, auparavant.

Autres possibilités d'utilisation des nombres

Essayons quelque chose d'un peu plus compliqué :

```
PRINT 25 + 39 {E}
```

Nous obtenons :

```
64
```

Nous négligerons désormais le OK. Sachez néanmoins qu'il apparaîtra à chaque fois sur la dernière ligne. « + » est appelé fonction. Il y a un grand nombre de fonctions en Basic, et, associées à des nombres, elles forment des expressions, ainsi :

```
1
```

est une expression formée d'un seul nombre, et :

```
25 + 39
```

est également une expression, mais composée de deux

nombres et de la fonction + . Quand l'ordinateur rencontre une telle expression dans une instruction de ce type, il la résout ; en d'autres termes, il fait tous les calculs et affiche le résultat.

L'expression suivante est très légèrement différente.

Essayez :

```
PRINT 25 + 39  {E}
```

Le résultat est le même que le précédent. De toute évidence, l'ordinateur ne se préoccupe pas des espaces que nous pourrions laisser entre les nombres et les fonctions. Nous utiliserons des espaces simplement pour nous permettre de lire plus aisément les instructions. Nous pouvons aussi les supprimer après PRINT, si nous le désirons. Mais nous ne pouvons intercaler un espace entre les lettres d'un mot clé : PR INT ne fonctionne pas.

Essayons d'autres expressions. Cette fois nous allons disposer les sorties (en d'autres termes ce qui s'inscrit sur l'écran) à la droite de nos entrées (ce que nous tapons sur le clavier).

Nous	*L'ordinateur*
PRINT 3 + 4 + 5 {E}	12
PRINT – 3 – 4 – 5 {E}	– 12
PRINT 3 – 4 – 5 {E}	– 6
PRINT 3.5 + 4.6 {E}	8.1
PRINT 4*6 {E}	24
PRINT 5/2 {E}	2.5
PRINT – 5/2 {E}	– 2.5

Ces exemples montrent ce que sont les fonctions arithmétiques fondamentales du Basic. Comme vous le voyez, les symboles sont peu différents de ceux que l'on emploie en arithmétique. Ce sont :

+	l'addition
–	la soustraction
*	la multiplication
/	la division

La plupart des ordinateurs ont beaucoup d'autres fonctions mathématiques. Le nôtre, par exemple, peut résoudre les expressions suivantes.

Essayez :

```
PRINT SQR(25) {E}
```

Essayez :

```
PRINT COS(45) {E}
```

Nous avons d'abord obtenu la racine carrée de 25 (5, au cas où vos souvenirs en mathématique se seraient un peu estompés) et ensuite, le cosinus de 45 (0,707106781). Le Basic de certains ordinateurs ne comporte pas ces commandes. Soyez rassuré, nous n'y aurons plus recours.

Il existe beaucoup d'autres règles particulières au Basic pour les calculs, par exemple, l'ordre dans lequel l'ordinateur résout de telles expressions :

```
25 * 4 + 23/96 * (2 + 3)
```

Nous ne les examinerons pas en détail, et vous n'aurez pas souvent à les manipuler. Si les calculs vous intéressent, vous trouverez de plus amples explications dans votre manuel d'utilisation ou dans un ouvrage sur le Basic plus orienté vers les mathématiques.

Maintenant, c'est à vous

Tapez vos commandes afin que l'ordinateur puisse exécuter ce qui suit. Si vous possédez un ordinateur, alors donnez vos réponses et vérifiez-en l'exactitude. Si vous n'en avez pas, vous trouverez p. 185 les réponses obtenues sur notre ordinateur. Mais vous devez toujours être conscient que pour ces exercices et ceux qui suivent, il n'y a pas qu'une seule réponse exacte. Par exemple, vous pouvez très bien laisser des espaces là où nous ne l'avons pas fait — et vice versa — et obtenir une instruction que l'ordinateur accepte. La seule manière de mettre à l'épreuve vos instructions est de les essayer sur votre ordinateur.

Demandez à l'ordinateur d'effectuer :

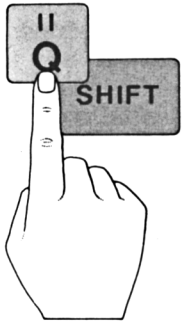
1) une addition : 56,4 plus 76,3
2) une division : 74 par 13,1
3) une multiplication : 45 par 7
4) une soustraction : 1025 moins 96

L'ordinateur et les lettres

La programmation ne fait pas seulement appel aux mathématiques. Avançons un peu, et voyons comment l'ordinateur manipule les lettres. Pour qu'une lettre soit imprimée, il est nécessaire de la placer entre guillemets, comme ceci :

Essayez :

```
PRINT "A"
```



Pour imprimer les guillemets
Sur la plupart des claviers, pour imprimer les guillemets, vous devez appuyer à la fois sur la touche SHIFT et sur celle qui représente les guillemets.

L'ordinateur imprime :

```
A
```

Mais, si vous oubliez les guillemets, il ne le fera pas.

Essayez :

```
PRINT A
```

Qu'a-t-il imprimé ? Probablement 0 (comme le nôtre) ou un message d'erreur, car l'ordinateur obéit à des règles spéciales qui lui indiquent ce qu'il doit faire pour chaque lettre que vous tapez. Dans une instruction comme :

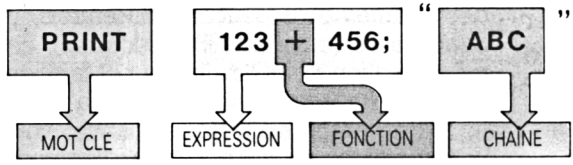
```
PRINT A
```

A revêt une certaine signification (que nous éclaircirons bientôt). Les guillemets sont donc là pour indiquer à l'ordinateur que la lettre (ou le groupe de lettres) ne revêt pas de signification particulière.

Essayez :

Nous	*L'ordinateur*
`PRINT "N" {E}`	`N`
`PRINT "NOIR" {E}`	`NOIR`
`PRINT "NOIR ET BLANC" {E}`	`NOIR ET BLANC`
`PRINT "  A DROITE" {E}`	`  A DROITE`
`PRINT  "A GAUCHE" {E}`	`A GAUCHE`
`PRINT "  " "DROITE" {E}`	`  DROITE`

Instructions en Basic



Une instruction en Basic se compose d'un mot-clé généralement suivi, soit de caractères qui forment une expression, soit de caractères entre guillemets, appelés chaînes, soit des deux. Dans cette instruction, l'expression est formée de deux nombres et d'une fonction (le signe +).

Nous découvrons ainsi une nouvelle règle. Quand nous plaçons un espace entre les guillemets, l'ordinateur en tient compte ; en revanche, il ignore les blancs, si ceux-ci se situent en dehors des guillemets.

En jargon informatique, les expressions entre guillemets sont appelées chaînes. Les chaînes peuvent contenir des lettres, comme dans l'exemple ci-dessus, des nombres, des espaces, des signes de ponctuation ou d'autres symboles entrés à l'aide du clavier, ou d'une autre manière.

Essayez tout ceci :

Nous	*L'ordinateur*
PRINT "12345" {E}	12345
PRINT "12 + 345" {E}	12 + 345
PRINT "12 – 345" {E}	12 – 345
PRINT "12 – 345 =" 12 – 345 {E}	12 – 345 = – 333
PRINT "POMME,POIRE" {E}	POMME,POIRE
PRINT "..,,;;;;;;???" {E}	..,,;;;;;;???

Avez-vous remarqué que l'ordinateur n'a pas calculé les expressions en additionnant ou en soustrayant, comme il le faisait quand nous utilisions les nombres hors des guillemets ? Dans le quatrième exemple nous avons procédé de telle sorte que l'ordinateur imprime, et l'expression, et le résultat (sur certains appareils, il faut placer un point-virgule après le second groupe de guillemets). Nous aborderons l'usage de point-virgule hors guillemets à la page 26. Avez-vous également remarqué que, cette fois, le premier espace devant les nombres positifs a été supprimé ?

Quelle est la longueur d'une chaîne ?

A chaque ordinateur correspond une longueur limite des chaînes qu'il peut traiter. Parfois, elle est réellement basse (environ 255 caractères), ce qui facilite la mémorisation pour l'ordinateur. Mais, quelquefois, une chaîne peut occuper tout l'espace mémoire (des milliers de caractères). Votre manuel vous dira ce que votre ordinateur peut accepter. Néanmoins, les longues chaînes sont rarement pratiques ; il est plus simple de les diviser en séries de chaînes courtes. Nous n'utiliserons pas de chaînes supérieures à 64 caractères dans ce livre. Généralement, il y a aussi une longueur limite pour les instructions que vous pouvez entrer avant d'enfoncer la touche ENTER. Bien que cela puisse surprendre, elle est souvent inférieure à la longueur de chaîne maximale. Il n'est pas toujours possible de déterminer cette dernière à partir d'une instruction unique. Mais nous pouvons, en revanche, savoir si la longueur maximale pour une instruction est susceptible de vous gêner. Il suffit pour cela d'entrer une instruction qui contient une longue chaîne.

Essayez :

```
PRINT " VOICI UN TEST QUI VA ME P
ERMETTRE DE SAVOIR SI MON ORDINA
TEUR ACCEPTE DE LONGUES INSTRUCT
IONS. JE NE M'ARRÊTERAI QUE LORSQ
QUE J'AURAI ATTEINT UNE LONGUEUR
D'INSTRUCTION QUI ME PARAÎTRA SU
FFISANTE" {E}
```

Nous espérons que vous n'avez pas appuyé sur ENTER avant que nous ne vous l'ayons dit ! Vous avez sans doute remarqué qu'il est inutile d'enfoncer cette touche en fin de ligne — alors que vous auriez dû presser la touche du retour chariot sur une machine à écrire. L'ordinateur débute de lui-même une nouvelle ligne. Il est très irritant de constater que si vous vous trouvez au milieu d'un mot, il s'en moque totalement et ne le reporte pas sur la ligne suivante dans sa totalité. Comme vous ne pouvez pas effectuer de retour chariot en appuyant sur ENTER sans provoquer l'exécution de l'instruction, vous devez vous accommoder du mot coupé tel qu'il est, ou taper une série d'espaces quand vous approchez de la fin de la ligne sur l'écran. Notre

ligne contient 32 caractères. Si la vôtre est de longueur différente, alors la coupure se fera bien évidemment en un point différent.

Que s'est-il passé ? Votre ordinateur vous a peut-être laissé taper tout ce texte et l'a imprimé sans rechigner. Ou alors il a tout accepté, mais il vous a donné un message d'erreur, au lieu d'imprimer le texte. Dans ces deux cas, votre manuel vous indiquera la longueur d'instruction maximale dont vous disposez. Plus probablement, il vous a laissé taper, disons, les deux premières lignes, puis s'est arrêté net.

Évidemment, il est très pratique qu'un ordinateur accepte de longues instructions, mais pour le but que nous nous sommes fixé, ses performances en la matière importent peu. Nous utiliserons des instructions ne dépassant pas 64 caractères, qui sont acceptées par presque tous les ordinateurs.

Lorsque l'écran affiche la chaîne, les coupures des mots n'apparaissent pas aux mêmes endroits que lorsque vous l'avez saisie, car l'ordinateur n'imprime pas l'instruction « PRINT ». Voici ce que vous obtenez :

```
VOICI UN TEST QUI VA ME PERMETTR
E DE SAVOIR SI MON ORDINATEUR AC
CEPTE DE LONGUES INSTRUCTIONS. JE
NE M'ARRÊTERAI QUE LORSQUE J'AUR
AI ATTEINT UNE LONGUEUR D'INSTRU
CTION QUI ME PARAÎTRA SUFFISANTE
```

Si vous voulez que votre sortie se présente bien, il vous faut alors apporter des corrections, comme nous le verrons lorsque nous commencerons à écrire des programmes.

L'addition des chaînes

Ce qui est pratique avec les ordinateurs, c'est que la fonction addition s'applique aussi bien aux chaînes qu'aux nombres. Il arrive souvent que l'on veuille former de longues chaînes à partir de chaînes plus courtes, lorsque l'on fait du traitement de texte, par exemple.

Essayez :

```
PRINT "AUTO" + "ROUTE" {E}
```

Qu'est-ce que l'ordinateur imprime ? Bravo !

AUTOROUTE

Il y a ici un piège pour les imprudents :

PRINT "22" + "33" {E}

ne donne pas "55", mais le contenu des deux chaînes mises bout à bout :

2233

Alors réfléchissez à deux fois avant d'ajouter des chaînes de nombres !

Le curseur

Si vous ne l'avez pas encore fait, examinez avec attention le curseur. C'est un petit symbole (carré lumineux, lettre lumineuse, etc.) qui apparaît sur l'écran lorsqu'il vous appartient de taper quelque chose. Vous pouvez le faire aller et venir (cependant, pas toujours au-dessus de l'endroit où vous vous trouvez déjà) en utilisant les touches qui commandent son déplacement (cherchez ces touches sur notre clavier p. 183). Notez qu'elles comportent des flèches indiquant les directions dans lesquelles il peut être bougé.

Le curseur est très utile parce qu'il vous indique l'endroit où va apparaître ce que vous tapez (jusqu'à présent, c'était toujours sur la ligne suivante). Bien sûr vous pouvez facilement déplacer le curseur en appuyant sur les touches prévues à cet effet, mais vous risquez d'induire en erreur votre ordinateur et d'obtenir une sortie qui s'affiche en des endroits bizarres, souvent exactement là où vous ne le vouliez pas.

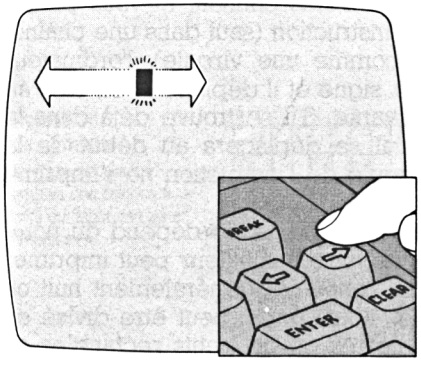
Certaines instructions peuvent déplacer le curseur sans qu'il imprime quoi que ce soit à l'écran.

Essayez :

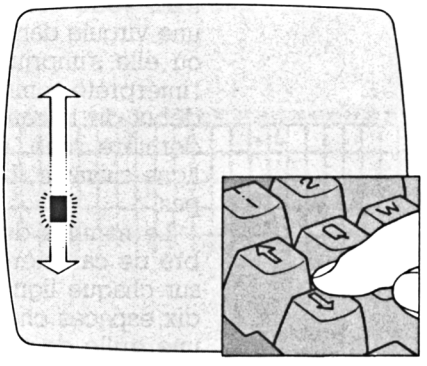
CLS {E}

Sur la plupart des ordinateurs (dont le nôtre), c'est la commande qui donne l'ordre à la machine :
— d'effacer l'écran
— de placer le curseur dans le coin supérieur gauche.

Comment bouger le curseur



Le curseur est le symbole lumineux, sur l'écran, qui vous indique à quel endroit va apparaître le caractère que vous êtes en train de taper. En appuyant sur les différentes touches du curseur, vous pouvez le faire bouger de haut en bas et de gauche à droite et ainsi, le positionner n'importe où sur l'écran.



L'instruction PRINT est inutile quand on tape CLS (effacer).

Certains ordinateurs ont des commandes différentes pour effacer l'écran et pour replacer le curseur en position initiale (en haut, coin gauche). D'autres ont une touche spéciale qui remplace la commande. Si CLS est inopérant sur votre ordinateur, consultez votre manuel qui propose sans doute une autre façon de procéder, ou bien consultez l'annexe. Nos notes sur les différentes machines vous aideront sûrement.

La ponctuation

Nous avons utilisé la virgule et le point dans les chaînes, mais toujours à l'intérieur des guillemets.

Essayons maintenant :

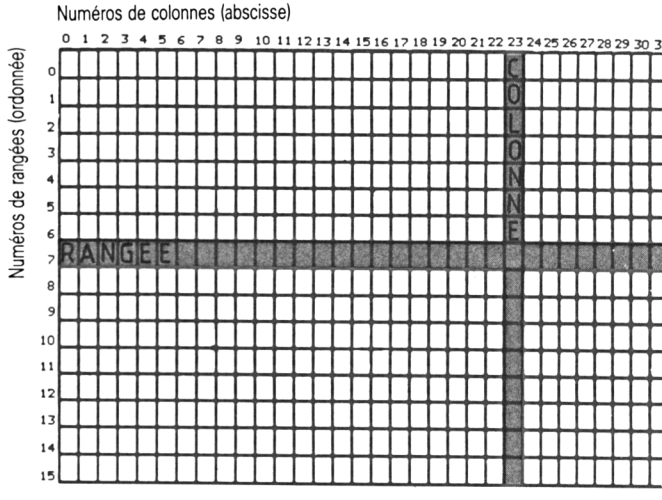
```
PRINT 1,2,3,4,5,6 {E}
```

Votre sortie devrait avoir une disposition semblable à celle-ci :

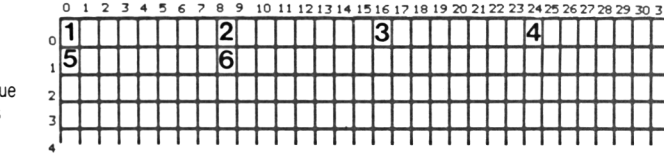
```
1       2       3       4
5       6
```

Pourquoi ? Il y a sur votre écran un certain nombre de « zones d'impression » ; vous ne pouvez pas les voir, mais votre ordinateur, lui, les connaît. Si vous placez une virgule dans une instruction (sauf dans une chaîne, où elle s'imprimera comme une virgule), l'ordinateur l'interprète comme un signe et il déplace le curseur au début de la zone suivante. S'il se trouve déjà dans la dernière zone, alors il se déplacera au début de la ligne suivante. La virgule de l'instruction ne s'imprime pas.

Le nombre de zones et d'espaces dépend du nombre de caractères que votre ordinateur peut imprimer sur chaque ligne. Les zones ont généralement huit ou dix espaces chacune. Votre écran peut être divisé en une grille de petits carrés ou de petits rectangles où viennent s'imprimer les caractères et sur lesquels se déplace le curseur. Une série de caractères imprimés horizontalement forme une rangée. Une série verticale forme une colonne. Nous pouvons assigner un numéro à chaque rangée et à chaque colonne. Sur notre écran de 32 colonnes, les zones d'impression commencent aux colonnes 0, 8, 16, 24.



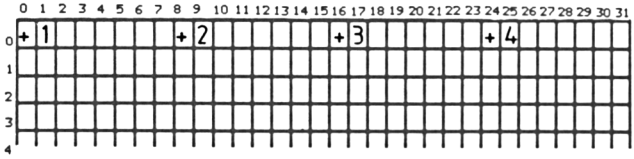
Grille d'écran
Cette grille peut être considérée comme une espèce de carte représentant la surface de l'écran.



Zones d'impression
Notre écran présente 4 zones sur lesquelles le curseur se déplace, lorsque l'on utilise la virgule dans les instructions.

Évidemment, quand nous avons demandé à l'ordinateur d'imprimer 1,2,3,4,5, il a commencé à la colonne 1 et non pas à la colonne 0, à cause du signe + invisible.

Impression des nombres
La virgule placera les nombres une colonne à droite du début des zones d'impression.



Notez qu'il est habituel, dans la numérotation des colonnes et des rangées, de commencer par zéro; ainsi, un écran de 32 colonnes et 16 rangées se terminera à la colonne 31 et à la rangée 15. Bien sûr, nous pouvons également utiliser cette technique pour les chaînes.

Essayons avec des chaînes courtes :

```
PRINT "ROUGE","VERT","BLEU","ROS
E" {E}
```

puis avec des chaînes plus longues :

```
PRINT "BLEU MARINE","ROUGE CERIS
E","ROSE CLAIR" {E}
```

Voici la sortie que vous devriez obtenir (mais tout dépend bien sûr de la taille des zones d'impression de votre écran).

```
ROUGE          VERT                BLEU          ROSE
BLEU MARINE                        ROUGE  CERISE
ROSE CLAIR
```

Essayons quelque chose de légèrement différent :

```
PRINT 1;2;3;4;5;6 {E}
```

Nous obtenons :

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

L'ordinateur imprime effectivement tous les nombres séparés d'un seul espace ; mais comme il utilise le signe + invisible, si ces nombres sont positifs, il s'intercale un espace supplémentaire entre chacun d'eux. La même version de cet exemple, mais avec un signe négatif, vous en convaincra :

Essayez :

```
PRINT -1;-2;-3;-4;-5;-6 {E}
```

Cette fois, nous obtenons :

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

Nous pouvons également utiliser les points-virgules avec des chaînes.

Essayez :

```
PRINT "ROUGE";"ORANGE";"BLEU";"R
OSE" {E}
```

Cette fois, pas d'espaces. Nous obtenons :

```
ROUGEORANGEBLEUROSE
```

Maintenant, c'est à vous

Demandez à votre ordinateur d'effectuer les tâches suivantes. Encore une fois, vous trouverez p. 185 les instructions qui permettent au nôtre de les réaliser.

5) Imprimez votre nom en haut et à droite de l'écran (conseil : mettez des espaces dans la chaîne).

6) Imprimez votre nom de haut en bas, en colonne (conseil : utilisez beaucoup de virgules).

7) Imprimez ROUGE, ORANGE, BLEU et ROSE avec un seul espace entre chaque mot, en utilisant des points-virgules.

Comment séparer les éléments imprimés ?

Faut-il introduire un signe de ponctuation entre les éléments que vous allez imprimer, s'il y en a plusieurs qui suivent le mot clé PRINT ?

Essayez :

```
PRINT "ROUGE" "JAUNE" "BLEU" "ROSE"  {E}
```

Notre ordinateur n'accepte pas cette instruction, cependant certains l'acceptent.

Essayez :

```
PRINT "ROUGE"1"ORANGE"2"ROSE"3"B
LEU"4 {E}
```

Cette fois-ci encore, notre ordinateur refuse l'instruction, le vôtre l'accepte peut-être. Et pourtant notre ordinateur accepte les nombres et les chaînes après PRINT.

Essayez :

```
PRINT "ROUGE";1;"ORANGE";2;"ROSE
";3;"BLEU";4 {E}
```

La sortie se présente ainsi :

```
ROUGE 1 ORANGE 2 ROSE 3 BLEU 4
```

Il est presque certain que votre ordinateur acceptera également cela. Sinon, vous découvrirez bientôt comment obtenir le même résultat avec des instructions différentes.

Les deux-points

La plupart du temps, à part le point-virgule, il n'y a qu'un signe de ponctuation qui se charge d'un sens particulier après la commande PRINT. Ce sont les deux-points « : ». Vous pouvez essayer d'autres signes, mais vous ne ferez qu'étendre vos connaissances en matière de messages d'erreur !

Essayez :

```
PRINT "ROUGE" : "ORANGE" : "VERT" : "V
IOLET" {E}
```

Nous obtenons :

```
ROUGE
?SN ERROR
```

(Vous vous souvenez ? C'est un message d'erreur.) Où sont passés ORANGE, VERT et VIOLET ? L'ordinateur

n'en a pas tenu compte parce que le signe : indique la fin d'une instruction en Basic. Les mots clé ne fonctionnent qu'à l'intérieur d'une seule instruction, aussi le mot clé PRINT n'opère-t-il plus au-delà des deux points. L'ordinateur a produit un message d'erreur car il se soucie peu d'obéir au mot clé ORANGE. Sur beaucoup d'ordinateurs, les deux-points vous permettent de placer plus d'une instruction sur une seule ligne de Basic (c'est-à-dire qu'il est tout à fait possible de saisir un texte dont la longueur dépasse une simple ligne, avant de presser la touche ENTER).

Essayez :

```
PRINT "ROUGE" PRINT "ORANGE":PRINT "VERT" {E}
```

Cette fois, notre ordinateur imprime les trois éléments de cette manière :

```
ROUGE
ORANGE
VERT
```

Le vôtre aussi ? S'il ne le fait pas, c'est qu'il ne peut utiliser qu'une seule instruction par ligne. Cela ne vous créera pas de difficulté puisque, lorsque nous aborderons la programmation, vous pourrez tout aussi bien vous débrouiller de cette façon.
La longueur maximale de l'instruction que nous avons déterminée page 20, correspond en fait à la longueur de ligne maximale. Si vous placez plusieurs instructions sur une ligne, leur longueur totale devra être inférieure à la longueur maximale d'une instruction.

Étudions maintenant un autre emploi du point-virgule, en le plaçant à la fin d'une instruction :

Essayez :

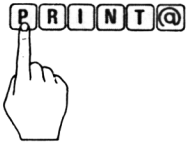
```
PRINT "UNE";:PRINT "GOUTTE" {E}
```

Après la première instruction, le curseur ne revient pas au début de la ligne suivante, comme il le fait normalement. Nous obtenons donc :

```
UNE GOUTTE
```

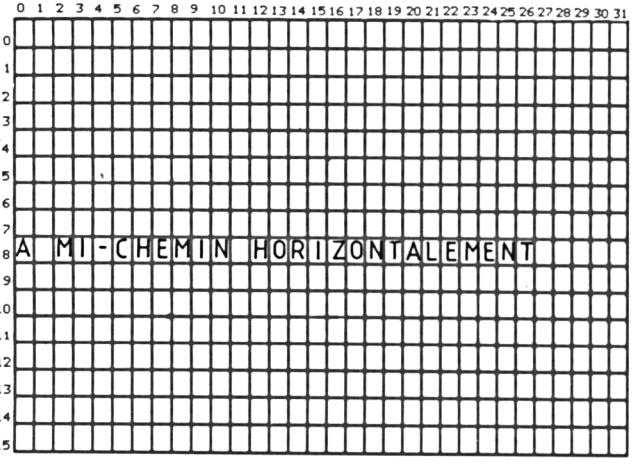
Présentation des sorties sur l'écran

Comme vous l'avez vu, nous pouvons utiliser les signes de ponctuation pour obtenir de l'ordinateur qu'il fasse ses affichages sur l'écran là où nous le désirons. Néanmoins, cette méthode ne permet pas de dessiner des figures et des images un peu complexes. En voici une plus simple, et qui offre plus de possibilités : composez d'abord, CLS pour effacer l'écran ; ensuite,



Essayez :

```
PRINT@ (8,0), "A MI-CHEMIN HORIZ
ONTALEMENT" {E}
```



Imprimer à un emplacement précis
Voilà comment notre écran se présente, après l'entrée de l'instruction PRINT @ ci-dessus (pour être plus clair nous avons omis l'instruction elle-même et le OK).

Notre ordinateur a accepté l'instruction et l'a effectuée (cf. résultat sur la figure). Votre résultat est-il différent ? Consultez les pages 172-174 pour savoir quelle est la commande équivalente sur votre micro-ordinateur. Savoir déterminer l'endroit où votre sortie apparaîtra grâce à PRINT @ est si utile, que dans le cas où cette commande serait inopérante sur votre ordinateur, il ne faut pas hésiter à consacrer du temps à rechercher une autre façon de procéder.

Vous avez peut-être deviné que les nombres entre parenthèses indiquent à l'ordinateur l'endroit où il doit procéder à l'affichage. Dans notre version de Basic, le premier nombre représente la rangée et le second, la colonne. Ils sont exactement semblables aux nombres que nous avons utilisés plus haut pour les zones

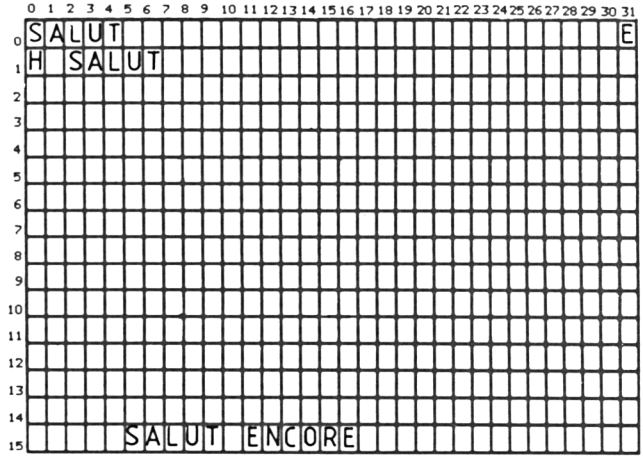
d'impression. L'instruction PRINT @ amène le curseur sur l'emplacement ainsi défini et déclenche l'affichage. (Dans notre Basic, la virgule qui suit la parenthèse de droite constitue une partie essentielle de l'instruction ; elle ne déplace pas le curseur vers la zone d'impression suivante, comme elle le ferait normalement).

Entrons un peu plus dans les détails. Sans vous reporter à la figure suivante, où pensez-vous que les instructions suivantes vont afficher leurs messages ?

```
PRINT@ (0,0), "SALUT"
PRINT@ (0,31), "EH! SALUT"
PRINT@ (15,5), "SALUT ENCORE"
```

Pour vous aider, regardez de nouveau le schéma de la page 24 ou celui qui se trouve ci-dessus. Vous pouvez voir que l'écran y est représenté par un plan quadrillé reproduisant 16 rangées et 32 colonnes. Consultez le manuel de notre ordinateur et vous y trouverez certainement un tableau semblable. S'il comporte un nombre différent de rangées et de colonnes, il vous faudra l'utiliser à l'exclusion de nos figures pour mettre en forme les sorties écran.

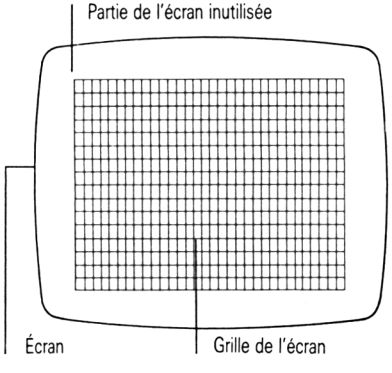
Autre exemple d'affichage précis
Cette figure illustre l'exemple ci-dessus.



Remarquez que lorsque nous demandons à notre ordinateur de placer le curseur à la colonne 31, il exécute fidèlement l'ordre qui lui est donné et imprime les lettres restantes au début de la ligne suivante. Si nous avions voulu imprimer « EH! SALUT » totalement à la droite de l'écran, où aurions-nous dû placer le curseur ? Utilisez votre tableau pour trouver la réponse.

Il se peut que votre manuel vous fournisse plusieurs tableaux, comprenant chacun un nombre distinct de rangées et de colonnes. Ces tableaux vous indiquent les différentes façons d'utiliser l'écran de votre ordinateur. Pour afficher des caractères, il ne vous sera normalement pas possible d'utiliser le tableau comportant le plus grand nombre de cases. Aussi, assurez-vous que vous avez choisi celui qui convient, si votre ordinateur vous en propose plusieurs. Vérifiez que votre ordinateur fonctionne effectivement sur la base du nombre de cases indiquées. Remplissez l'écran de caractères et comptez les rangées et les colonnes ! Comme ceci est votre premier ouvrage consacré au Basic, nous nous bornerons à utiliser la définition de notre ordinateur (16 × 32).

Au fait, vous remarquerez probablement que les sorties produites par l'ordinateur, et que le tableau représentant votre écran, ne couvrent pas complètement la surface de ce dernier. C'est que l'image subit des distorsions sur le bord de l'écran : l'ordinateur ne l'utilise donc pas pour l'affichage du texte. Si c'est un ordinateur disposant de la couleur, il est sûrement muni d'un ensemble de commandes spéciales qui permettent de changer la couleur du bord, afin d'éviter cette distorsion.



Surface utile de l'écran
La plupart des ordinateurs ne se servent que de la partie centrale de l'écran, laissant ainsi une bordure inutilisée, qu'on peut colorier.

Dans les boutiques de micro-informatique, vous pouvez acheter des blocs de feuilles spéciales qui représentent l'écran vierge sous forme d'un tableau. Vous pouvez les utiliser pour prévoir la disposition de vos sorties lors de la rédaction de votre programme. Cela se révèle très utile pour les instructions de positionnement. En fait, vous pourriez utiliser l'une de ces feuilles dès maintenant, car nous allons dessiner quelque chose sur l'écran.

Pour dessiner une image

Grâce à l'instruction PRINT @, ainsi qu'aux règles de ponctuation que nous avons étudiées, nous allons dessiner quelques images très simples.
Nous tiendrons compte dans la plupart de nos exemples du nombre limite de caractères pour une instruction, ou pour une série d'instructions juxtaposées (64).
Vous pouvez, bien sûr, créer une image avec plusieurs séries d'instructions, mais comme vous ne pouvez pas effacer l'écran après chacune d'elles sans perdre l'image qui vient d'apparaître, vous risquez de voir cette dernière se superposer à la succession de vos instructions.

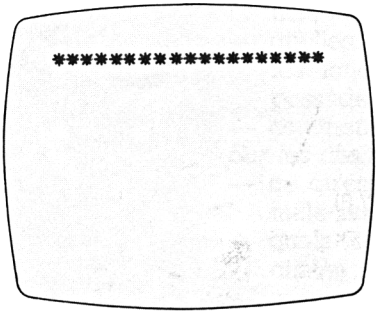
Voici quelques-unes des possibilités offertes. Rappelez-vous que l'ordinateur ne prend pas en considération les espaces qui ne se trouvent pas entre guillemets; vous pouvez donc toujours les éliminer : ils contribuent, en effet, à vous rapprocher du total fatidique des 64 caractères.

a) Imprimez une rangée d'astérisques sur toute la longueur du bord supérieur de l'écran. Vous devez utiliser une longue chaîne qui les contiendra tous. N'oubliez pas les guillemets.

Essayez :

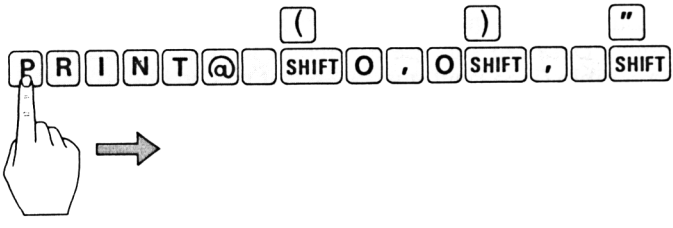
```
CLS: PRINT@ (0,0), "************
********************" {E}
```

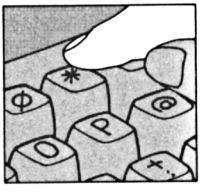
Nous découvrirons plus loin d'autres méthodes permettant d'arriver plus aisément à ce résultat. Notez bien la façon dont nous avons combiné deux instructions, l'une introduite par la commande CLS, l'autre par le mot clé PRINT.



Pour dessiner

Grâce à l'instruction PRINT @, on peut positionner ses sorties écran là où on le désire. PRINT @ peut, par conséquent, être utilisé pour créer des formes simples ou pour « décorer » l'écran. Dans cet exemple, nous avons d'abord effacé l'écran à l'aide de la commande CLS et, ensuite, positionné les astérisques contenus dans une chaîne à l'endroit désiré.





b) Divisez l'écran en deux parties par une ligne horizontale.

Essayez :

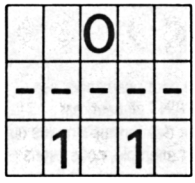
```
CLS: PRINT@ (13,0), "-----------
----------" {E}
```

c) Utilisez le chiffre 1 pour dessiner une ligne verticale. C'est un peu plus amusant.

Essayez :

```
CLS: PRINT,,1,,,,1,,,,1,,,,1,,,
,1,,,,1,,,,1,,,,1,,,,1,,,,1 {E}
```

La lettre « I » serait plus flatteuse, mais il faudrait l'inclure dans une chaîne, et pour tracer une ligne verticale jusqu'en bas de l'écran, 64 caractères ne suffiraient pas.



d) Dessinez un petit bonhomme en vous servant de lettres pour représenter les bras, les jambes, etc., comme cela :

```
  0
-----
 1 1
```

```
CLS:PRINT@(6,8),"0":PRINT@(7,6),
"-----":PRINT@(8,7),"1 1" {E}
```

Maintenant, c'est à vous

Essayez de faire réaliser à votre ordinateur, ou au nôtre, tout ce qui suit.

8) Dessinez un petit chien, le plus ressemblant possible (il se peut que ce ne soit pas un chef-d'œuvre, mais faites au mieux), en haut et à droite de l'écran.

9) Effacez et écrivez maintenant votre nom au milieu de l'écran.

10) Tracez une diagonale (en utilisant le symbole /) du coin supérieur droit au coin inférieur gauche (cela nécessite plus de 64 caractères).

Contrôle

A la fin de chaque chapitre, nous résumerons ce que, nous l'espérons, vous aurez appris. Si votre ordinateur est différent du nôtre, pour obtenir certains renseignements, vous devrez consulter l'annexe 1, ou votre propre manuel. Assurez-vous que vous avez parfaitement assimilé la façon dont le Basic traite les points qui vous sont exposés ci-dessous, avant de passer au chapitre suivant.

Arrivé à la fin de ce chapitre, vous devriez savoir :

Sur les instructions Basic

— comment se présente une instruction simple en Basic ;
— la longueur maximale d'instruction acceptée par votre ordinateur ;
— si votre ordinateur accepte plus d'une instruction sur une seule ligne Basic ;
— comment utiliser la touche ENTER, ou son équivalent sur votre ordinateur, pour valider une instruction ;
— comment utiliser les deux-points pour séparer les instructions.

Sur les fonctions et les expressions

— quels sont les symboles pour l'addition, la multiplication, la division, la soustraction ;
— quelles sont, pour les plus communes d'entre elles, les fonctions arithmétiques que votre ordinateur possède ;
— comment se présente une expression simple.

Sur les chaînes

— ce qu'est une chaîne, quelle est sa longueur maximale sur votre ordinateur ;
— quels caractères vous pouvez mettre dans une chaîne ;
— comment utiliser la fonction « + » avec des chaînes.

Sur l'écran

— combien de lignes et de colonnes votre écran possède ;
— comment utiliser l'instruction PRINT,
  — pour imprimer des nombres ;
  — pour imprimer d'autres caractères ;
  — avec des blancs, des points-virgules, des virgules ;
  — pour imprimer une succession d'éléments différents ;
— comment effacer l'écran à l'aide de CLS ou de la commande correspondante ;
— comment utiliser le curseur ;
— comment disposer vos sorties en utilisant le mot clé PRINT @ (ou correspondant).

2

Premiers Programmes

Au cours de ce chapitre, nous étudierons :
— ce que veut dire : écrire un programme,
— les commandes : RUN, LIST, NEW, SAVE,
— les instructions : END, LOAD, REM,
— l'instruction : GOTO,
— les caractères graphiques.

Dans le précédent chapitre, nous avons exploré les possibilités du mot clé PRINT et nous avons vu comment nous pouvions l'utiliser dès la mise en marche de l'ordinateur. Nous avons réalisé un certain nombre de choses grâce à lui, mais nous n'avons pas, à proprement parler, écrit de programme. Alors que veut dire : écrire un programme, et en quoi cela diffère-t-il de ce que nous avons déjà fait ?

Quand nous utilisons l'ordinateur en mode direct, comme nous l'avons fait dans le chapitre précédent, l'ordinateur exécute notre instruction dès que nous appuyons sur ENTER. Ce n'est pas le cas dans le mode programme ; au lieu de passer immédiatement à l'exécution, il nous laisse élaborer une série d'instructions qui finissent par former un programme, que nous pourrons ensuite exécuter, lister, garder sur une cassette, ou, tout simplement, supprimer.

Comment l'ordinateur fait-il la différence entre le mode direct et le mode programme ? Nous lui indiquons que nous programmons en affectant un nombre (appelé numéro de ligne) à chaque ligne d'instruction qui compose une partie de notre programme.

Pour rendre les choses plus claires, passons à un petit exemple.

Essayez :

```
10   PRINT "CECI" {E}
20   PRINT "EST" {E}
30   PRINT "UN" {E}
40   PRINT "PROGRAMME" {E}
50   END {E}
```

Pour ce programme et pour les suivants, nous nous en tiendrons à des instructions qui utilisent le mot clé PRINT. Ainsi, nous prêterons plus d'attention à la structure du programme qu'à ce qu'il contient.

Comme vous pouvez le voir, ce programme simple a cinq lignes. Les nombres, sur la gauche, sont les numéros de ligne; nous parlerons bientôt de ces derniers plus longuement, et nous expliquerons pourquoi nous choisissons les nombres 10, 20, 30, etc., au lieu de 1, 2, 3. Les lignes 10 à 40 contiennent de simples instructions d'affichage, et la ligne 50 fait apparaître un nouveau mot clé : END. END permet à l'ordinateur de savoir qu'il a atteint la fin du programme. Vous pensez sans doute que ce devrait être évident pour l'ordinateur. Dans de nombreux cas, c'est vrai, car il n'y a plus de lignes à exécuter. Sur certains ordinateurs, il n'est alors pas nécessaire d'utiliser END. Le nôtre, cependant, s'attend à trouver END à la fin de chaque programme.

Revenons à notre programme. Nous le tapons tel qu'il vous apparaît ici, en enfonçant la touche ENTER à la fin de chaque ligne, comme nous l'avons indiqué plus haut. Au fur et à mesure que l'on tape ces lignes, elles apparaissent sur l'écran de la même manière que lorsque l'on travaille en mode direct, mais il ne se passe rien. Comment faire exécuter le programme? En recourant à une commande spéciale.

Essayez :

RUN {E}

L'ordinateur imprime maintenant ce que nous lui avons demandé. Voici comment devrait se présenter notre écran :

```
10 PRINT "CECI"
20 PRINT "EST"
30 PRINT "UN"
40 PRINT "PROGRAMME"
50 END
RUN
CECI
EST
UN
PROGRAMME
OK
```

Les mots s'affichent les uns en-dessous des autres, conformément à ce que nous avions demandé. Que se serait-il passé si nous avions ajouté un point-virgule à la fin de chaque ligne? Bravo! Le curseur n'aurait pas changé de ligne et l'ordinateur aurait imprimé tous les mots, les uns à la suite des autres, sur la même ligne (sans oublier, bien sûr, d'intercaler des espaces entre les mots de la chaîne). C'est ce que nous allons faire dans le programme suivant. Mais auparavant, finissons-en avec celui-ci.

Essayez :

LIST {E}

La liste complète des instructions devrait apparaître sur votre écran, avec les numéros de ligne. LIST est très utile. En effet, lorsque l'on entre une ligne, elle s'inscrit à la suite du texte qui remonte d'un cran. Quand l'écran est rempli, les premières lignes disparaissent au fur et à mesure que vous en entrez de nouvelles. LIST vous permet de savoir ce que vous avez écrit au début d'un long programme, lorsque les premières lignes ont disparu de l'écran. Vous pouvez aussi utiliser LIST pour examiner votre programme dans son ensemble après l'avoir exécuté, si l'exécution l'a chassé de l'écran, ou si vous avez utilisé la commande CLS.

LIST a également d'autres applications, par exemple :

Essayez :

LIST 10 {E}

Seule la ligne 10 apparaît. Voici donc ce que vous devriez obtenir en bas de votre écran :

```
LIST 10 {E}
10 PRINT "CECI"
OK
```

Essayez :

LIST 20-50 {E}

Cette fois les lignes 20 à 50 apparaissent sur l'écran.

Essayez :

LIST -30 {E}

Vous devriez obtenir toutes les lignes du programme jusqu'à la ligne 30, incluse. Et, pour finir,

Essayez :

LIST 40- {E}

Ceci vous donne toutes les lignes, à partir de 40, jusqu'à la fin du programme.

Sur la plupart des ordinateurs, la commande LIST fait défiler la liste entière, ou la partie de liste désirée. Le défilement sera beaucoup trop rapide pour que vous puissiez voir les premières lignes d'un long programme, aussi, dans ce cas, vous devrez demander à l'ordinateur de lister le programme par groupes de lignes.

Après avoir utilisé la commande LIST, vous pouvez encore faire exécuter votre programme en composant RUN. Vous pouvez aussi effacer l'écran à l'aide de CLS et ensuite faire exécuter le programme. Bien que CLS efface ce qui est affiché sur l'écran, il n'altère pas ce qui est mis en mémoire. Alors, comment se débarrasser du programme ?

Essayez :

NEW {E}

Avec notre ordinateur, rien ne s'efface sur l'écran. Quel est donc l'effet de NEW ? Tapez RUN à nouveau : le programme a été effacé de la mémoire et rien ne se passe. Nous recourrons, en général, à NEW, chaque fois que nous en aurons terminé avec un programme, et que nous voudrons en commencer un autre, sinon l'ordinateur mélangera tout et ne saura plus quelle ligne appartient à quel programme. En utilisant NEW, nous sommes sûrs de n'avoir qu'un seul programme en mémoire...

Maintenant que nous avons utilisé NEW, nous pouvons passer à un autre programme.

Essayez :

```
1 PRINT "NOTATION" {E}
2 PRINT "DES NUMÉROS"; {E}
3 PRINT "DE LIGNE." {E}
4 END {E}
```

C'est le dernier exemple où nous vous rappelons qu'il vous faut composer ENTER à la fin de chaque ligne. Par la suite, nous espérons que vous vous en souviendrez.

Vous remarquerez que, comme promis, nous avons utilisé une ponctuation différente dans ce programme. Comment se fera l'affichage lorsque vous appuierez sur RUN ? Comme cela :

```
NOTATION DES NUMÉROS DE LIGNE.
```

Notre but principal était de vous amener à réfléchir sur les numéros de ligne. La dernière fois, nous avions utilisé 10, 20, 30 etc. Cette fois-ci, nous avons opté pour 1, 2, 3, 4. Nous pouvons, dans la limite du raisonnable, choisir n'importe quels nombres pourvu qu'ils soient entiers, qu'ils ne dépassent pas une certaine grandeur (de l'ordre de quelques mille, en général), et qu'ils soient ordonnés afin d'indiquer la succession des instructions du programme.

Il est d'usage d'employer des multiples de 10, car si l'on veut insérer une nouvelle ligne entre des lignes déjà existantes (que l'on ait oublié quelque chose ou que l'on désire apporter une amélioration, ou rajouter certains détails), on a alors le choix entre neuf numéros.

Mélanger les lignes

Autre avantage des numéros de ligne : votre ordinateur classera lui-même les instructions dans l'ordre. Vous n'avez pas à taper les lignes dans leur succession exacte. Bien sûr, c'est un bon entraînement que d'élaborer votre programme dans l'ordre logique, avant de le taper. Il reste que cette possibilité offerte par l'ordinateur est très commode et que si vous voulez ajouter quelque chose à votre programme, vous n'avez qu'à taper la nouvelle ligne et l'ordinateur la glissera là où il faut. De la même manière, si vous vous apercevez que vous avez commis une erreur dans une ligne, après l'avoir entrée, il suffit de la retaper correctement :

l'ordinateur effacera la première version et imprimera la seconde en bonne et due place.

Nous vous proposons maintenant un programme un peu trompeur, afin d'illustrer la façon dont l'ordinateur classe les lignes. Nous allons dessiner une image simple sur l'écran, mais, cette fois, il nous sera difficile de savoir à quoi elle ressemblera avant d'avoir exécuté le programme. Si l'image n'a pas l'air de se présenter correctement sur votre écran, ou si vous n'avez pas d'ordinateur, nous vous montrons à la page suivante ce que nous avions l'intention de vous faire exécuter.

Essayez :

```
50 PRINT "OO  O  OO"
70 PRINT "  O  O"
10 PRINT "  OOO"
90 END
40 PRINT "OO + + OO"
80 PRINT "  OOO"
30 PRINT "  OO  OO"
20 PRINT "  OOOOO"
60 PRINT "  O --- O"
```

Vous voyez que nous avons même une instruction END au milieu de notre listage. Cela ne fait rien, l'ordinateur ne s'en occupera qu'après avoir classé le programme. Alors, cette instruction prendra la place que nous lui avions assignée : à la fin.

Maintenant, tapez LIST : vous allez découvrir l'un des autres avantages de cette commande. Elle remet vos instructions dans le bon ordre, quel que soit celui dans lequel vous les avez tapées.

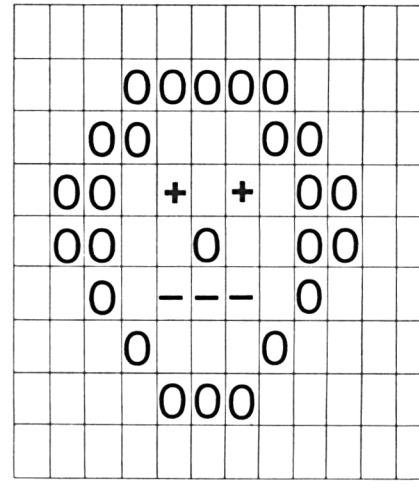
Maintenant, c'est à vous

Nous allons vous donner quatre programmes courts. Regardez-les bien. S'exécuteront-ils, et pourquoi ? Déterminez-en la sortie (réponses p. 185).

1)
```
10   PRINT: "SALUT LA COMPAGNIE"
20   END
```

2)
```
10   PRINT "MARIE JEANNE"
5    END
```

Programme « visage »
Voici l'exécution du programme dont nous avions mélangé les lignes, page 41. Le résultat vous surprend-il ? Si vous aviez listé les lignes avant d'exécuter le programme, vous auriez pu découvrir ce visage.



3)
```
30   END
25   PRINT "  1     1     1  "
10   PRINT "000   000   000"
20   PRINT "000   000   000"
29   PRINT "  1/    1/    1/ "
```

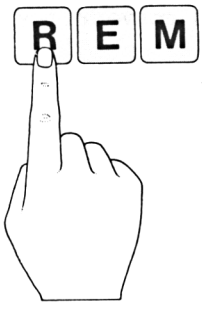
4)
```
1000  PRINT "AU REVOIR"
800   PRINT
700   PRINT
200   PRINT
300   PRINT "SALUT"
1010  END
```

L'instruction REM

Il est temps d'introduire un autre mot clé : REM. C'est l'un des nombreux mots clé que nous n'utilisons qu'en mode programme, jamais en mode direct. Voyons quelle en est l'utilité.

Essayez :

```
10   REM PROGRAMME PROGRAMME
20   REM CE PROGRAMME
30   REM N'A PAS L'AIR DE
40   REM FAIRE GRAND'CHOSE
50   PRINT "SALUT VOUS TOUS"
60   END
```



Que se passe-t-il ? Voici ce qui devrait apparaître à l'écran quand vous appuyez sur RUN.

```
SALUT VOUS TOUS
```

Vous savez d'où cela vient : de la ligne 50. Qu'est-il arrivé au texte qui précédait la ligne 50 ? Rien. En effet, REM demande à l'ordinateur d'ignorer totalement le texte qu'il introduit. Il en résulte que ce doit être la dernière instruction d'une ligne, ou la seule. Il n'est pas possible de se servir, sur la même ligne, d'un autre mot clé, après avoir employé REM.

Alors, quelle en est l'utilité ? Pourquoi taper des mots si l'ordinateur ne s'y intéresse pas ? Réponse : parce que vous, ils vous intéressent. En fait, les REMarques sont utilisées pour vous permettre d'introduire des rappels et des commentaires dans le listage du programme, soit pour vous-mêmes, soit pour les personnes qui utiliseront le programme.

Vous pourrez faire suivre REM, par exemple :
— du nom du programme. Nous en dirons quelques mots lorsque nous traiterons de la sauvegarde des programmes (leur préservation en mémoire) ;
— de votre propre nom ;
— de notes sur l'effet de chaque ligne du programme. Ceci est très important, car l'instruction REM devient alors un guide précieux, pour vous, ou pour l'utilisateur du programme.

Afin de donner une application concrète à ce que nous venons de voir, peut-être devrions-nous essayer de réaliser un programme long, mais peu compliqué. Nous allons dessiner des figures à l'aide de PRINT @ afin de vous rafraîchir la mémoire. Si nous essayions de dessiner une maison, à nouveau à l'aide de lettres ? Les instructions REM nous serviront à présenter les sections du programme qui correspondent aux différentes parties de la maison : murs, toits, portes, etc. Et pour bien les distinguer, nous commencerons chaque portion du programme par un numéro de ligne aisément mémorisable.

Munissez-vous d'une feuille de papier millimétré, ou de papier quadrillé pour ordinateur, ou alors dessinez un tableau quadrillé sur une simple feuille de papier. Ensuite, suivez bien nos instructions pour dessiner la maison. Nous nous attacherons à programmer dans les règles de l'art. Nous concevrons donc entièrement notre dessin avant de passer au codage du programme

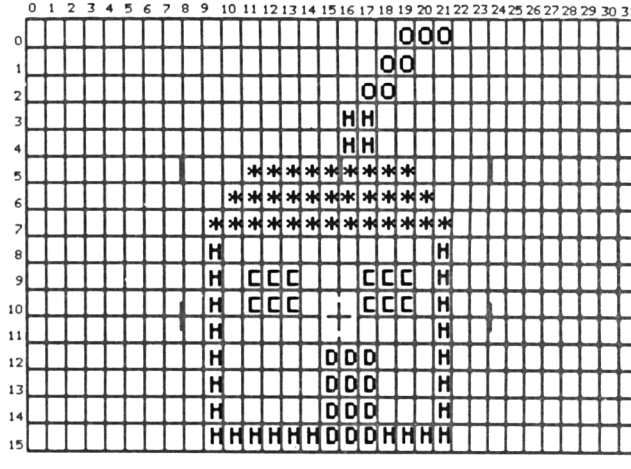
proprement dit (c'est-à-dire, avant de convertir nos idées en instructions Basic).

La figure vous présente la maison que nous allons dessiner. Nous n'avons pas la prétention d'être des artistes ; alors, si vous désirez apporter des modifications à ce schéma, ne vous gênez pas ! Les lignes et les colonnes ont été numérotées, afin de vous indiquer l'endroit où nous comptons placer notre maison. Encore une fois, vous pouvez adapter ce schéma à la taille de votre écran et le modifier selon vos préférences.

Travaillons dans un ordre logique afin que vous puissiez voir la maison se construire petit à petit. Nous dessinerons d'abord les murs : tapez les lignes qui suivent dès maintenant. Le programme n'est pas encore terminé, mais vous pouvez exécuter ces premières lignes. Vous obtiendrez peut-être un message d'erreur vous indiquant que l'instruction END fait défaut, mais vous pourrez voir ce qui s'affiche et serez en mesure de vérifier que vous n'avez pas commis de fautes de frappe.

Programme « maison »

Voici le dessin de la maison que nous allons réaliser. Il est représenté sur le tableau quadrillé reproduisant l'écran de notre ordinateur.



```
100 REM DESSIN DES MURS
110 PRINT@ (8,9), "H"
120 PRINT@ (8,21), "H"
130 PRINT@ (9,9), "H"
```

```
140 PRINT@ (9,21), "H"
150 PRINT@ (10,9), "H"
160 PRINT@ (10,21), "H"
170 PRINT@ (11,9), "H"
180 PRINT@ (11,21), "H"
190 PRINT@ (12,9), "H"
200 PRINT@ (12,21), "H"
210 PRINT@ (13,9), "H"
220 PRINT@ (13,21), "H"
230 PRINT@ (14,9), "H"
240 PRINT@ (14,21), "H"
250 PRINT@ (15,9), "HHHHHH     HHHH";
```

La dernière ligne comportait plus de 32 caractères, comme vous avez pu le constater si vous avez un écran possédant 32 colonnes d'affichage. Puisque, dans cet ouvrage, les lignes comportent plus de 32 caractères, nous adapterons la présentation de nos instructions en conséquence : nous ne reproduirons donc plus exactement la ligne de 32 caractères obtenue sur notre écran. De toute façon, ne vous tracassez pas, il est peu probable que nous utilisions des instructions d'une longueur telle que votre ordinateur ne puisse les accepter, même si chaque instruction couvre plus d'une ligne. Si néanmoins, elles s'avèrent trop longues, séparez-les en deux, et utilisez un autre numéro de ligne pour la deuxième partie. Bien sûr, assurez-vous que votre deuxième ligne s'imprime au bon endroit.

Soyez très attentifs à la ponctuation pour ce genre de dessin. Ainsi, le point-virgule est vraiment indispensable à la fin de la ligne 250. Sinon, le curseur en se déplaçant vers la ligne suivante dénaturerait votre dessin, en décalant l'affichage d'une rangée vers le haut.

La saisie de ces instructions est un travail fastidieux. En effet, vous ne connaissez pas encore les commandes qui permettraient de l'accélérer. Néanmoins, le reste du programme demandera un travail moins laborieux, puisque nous imprimerons plus de caractères par instruction.

Et maintenant, commençons par le toit :

```
300 REM DESSIN DU TOIT
310 PRINT@ (5,11), "*********"
320 PRINT@ (6,10), "**********"
330 PRINT@ (7,9), "**********"
```

Les fenêtres :

```
400 REM DESSIN DES FENÊTRES
410 PRINT@ (9,11), "CCC  CCC";
420 PRINT@ (10,11), "CCC  CCC";
```

Cette fois encore, les points-virgules placés à la fin de chaque ligne sont très importants. Sans eux, le mur de droite serait effacé quand le curseur passe à la ligne suivante.
La porte :

```
500 REM DESSIN DE LA PORTE
510 PRINT@ (12,15), "DDD";
520 PRINT@ (13,15), "DDD";
530 PRINT@ (14,15), "DDD";
540 PRINT@ (15,15), "DDD";
```

Et enfin, la cheminée et la fumée :

```
600 REM DESSIN DE LA CHEMINÉE ET DE LA PORTE
610 PRINT@ (4,16), "HH"
620 PRINT@ (3,16), "HH"
630 PRINT@ (2,17), "OO"
640 PRINT@ (1,18), "OO"
650 PRINT@ (0,19), "OOO"
```

Nous avons notre dessin, mais le programme n'est pas pour autant terminé. Ajoutez ces lignes :

```
10  REM***MAISON***
20  REM PAR SUSAN CURRAN
30  CLS
700 END
```

N'ayez pas de scrupules à inscrire votre propre nom en 20. Les astérisques à la ligne 10 ont simplement pour objet de faire ressortir le titre. Si vous utilisez un ordinateur, listez maintenant le programme dans son ensemble. Est-ce que vous voyez combien les REMarques facilitent la compréhension de chaque étape du programme ? Grâce à elles, il serait facile de changer une partie du dessin que vous n'aimez pas, par exemple, en utilisant la lettre F ou T au lieu de C pour les fenêtres. Comme nous l'avons déjà dit, pour procéder à une modification, il suffit de taper de nouvelles lignes. Elles remplaceront automatiquement les lignes précé-

dentes si vous les entrez avec les mêmes numéros de ligne. Pour supprimer une ligne sans la remplacer, il suffit de taper son numéro suivi d'ENTER, comme cela :

20 {E}

La ligne vide 20 se substituera à la précédente qui sera alors effacée. Lancez (RUN) maintenant le programme.

Pour sauvegarder vos programmes

Ce programme est assez ennuyeux en comparaison de ceux que nous effectuerons bientôt, mais pour vous encourager et vous aider à penser que vous avez produit quelque chose de concret, pourquoi ne pas le mettre en mémoire afin de l'utiliser à nouveau ? Si vous ne l'enregistrez pas sur une cassette ou sur une disquette, alors, dès que vous éteindrez l'ordinateur, le fruit de votre travail sera irrémédiablement perdu.

Notre système comporte un lecteur de cassettes que nous utilisons pour enregistrer nos programmes. Nous nous servons de la commande SAVE pour dire à l'ordinateur de mettre le programme en mémoire.

```
SAVE "MAISON"
```

Maintenant, l'ordinateur sait que notre programme s'appelle « MAISON » ! Lorsque nous tapons SAVE, suivi d'un mot entre guillemets quel qu'il soit, l'ordinateur sait que ce mot désigne le programme que nous venons d'entrer, et il mémorise ce dernier sous ce titre. Nous pouvons communiquer à l'ordinateur un nom différent de celui que nous avons utilisé dans l'instruction REM, au début du programme. Sur notre ordinateur, en effet, il faut limiter à huit le nombre de lettres que comportent les titres ; alors que l'on a toute latitude quant à la longueur des REMarques. Votre manuel devrait normalement vous indiquer la taille limite des titres de programmes acceptée par votre ordinateur.

La procédure de sauvegarde peut varier considérablement d'une machine à l'autre, nous vous laissons donc le soin de consulter votre manuel. Quand vous aurez sauvegardé votre programme, vous pourrez le charger (LOAD) à nouveau, toujours en suivant les instructions de votre manuel. Quand votre programme sera chargé, tapez RUN et l'ordinateur l'exécutera. Bien sûr, si vous n'avez pas éteint l'ordinateur, il est inutile de le charger pour l'exécuter, puisqu'il est toujours présent en mémoire.

Maintenant, c'est à vous

Élaborez de nouveaux programmes de dessins, en recourant abondamment aux instructions REM, afin de bien assimiler la façon dont on les construit. Donnez-leur un titre et, si vous le pouvez, testez-les sur votre ordinateur. Nous ne vous donnerons pas de solutions (nous allons bientôt découvrir des méthodes plus simples pour produire des dessins — le chapitre 8 contient un certain nombre d'exemples intéressants), mais vous trouverez ci-dessous des suggestions de programmes que vous pouvez essayer de mettre en forme.

5) une voiture
6) un vaisseau spatial
7) un jardin fleuri
8) un éléphant

L'instruction GOTO

Comme vous l'avez sûrement découvert en effectuant le programme « MAISON », la dernière partie de l'exécution peut gâcher l'effet escompté. L'ordinateur se fait un devoir de vous indiquer qu'il a terminé en affichant OK (ou un message équivalent) et, sauf si vous êtes vigilants, il le placera au beau milieu de votre œuvre. Comment l'en empêcher? Il suffit de s'assurer qu'il n'atteindra jamais la fin du programme. La manière la plus simple est de le faire tourner en rond en lui faisant accomplir une instruction qui n'affecte pas le reste du programme. Pour ce faire, nous avons besoin d'un nouveau mot clé : GOTO. Comme nous l'avons vu, l'ordinateur exécute les instructions d'une ligne de programme, puis passe immédiatement à la ligne suivante, dans l'ordre croissant des numéros. GOTO permet de modifier cette façon de procéder en renvoyant directement l'ordinateur à la ligne que nous lui indiquons (l'une de celles qui composent le programme). Voici comment l'utiliser :

```
GOTO 10
```

demande à l'ordinateur d'exécuter les instructions de la ligne 10.

```
GOTO 500
```

demande à l'ordinateur d'exécuter les instructions de la ligne 500.

Il ne faut jamais demander à l'ordinateur de se rendre à une ligne qui n'existe pas dans le programme ; il ne s'y retrouverait plus !

Voici maintenant un petit programme illustrant le fonctionnement de GOTO.

Essayez :

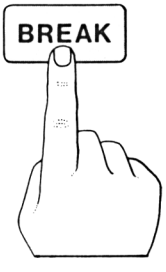
```
10 REM PROGRAMME GOTO
20 PRINT "CECI" : GOTO 70
30 PRINT "PROGRAMME" : GOTO 60
40 PRINT "STUPIDE" : END
50 PRINT "UN" : GOTO 30
60 PRINT "UN PEU" : GOTO 40
70 PRINT "EST" : GOTO 50
```

L'ordinateur suit nos instructions à la lettre et imprime le message suivant :

```
CECI
EST
UN
PROGRAMME
UN PEU
STUPIDE
```

Pour que l'ordinateur n'affiche plus rien, nous constituons une « boucle vide » qui lui demande, par exemple, de se rendre à la ligne où il se trouve déjà. Ainsi, au lieu de conclure notre programme « MAISON » par END en ligne 700, nous pourrions taper :

```
700 GOTO 700
```



Si votre programme « MAISON » se trouve encore en mémoire, testez-le avec cette nouvelle instruction. Faites bien attention d'assigner à la ligne contenant l'instruction GOTO le numéro que vous aviez donné à l'instruction END.

Lancez à nouveau le programme (RUN). Vous constatez que OK n'apparaît plus.

Bien sûr, le programme va s'exécuter indéfiniment, il nous faut donc trouver un moyen pour l'arrêter. Sur notre appareil, il suffit d'enfoncer la touche BREAK. Sur d'autres, la touche STOP remplit le même office.

Constituer une boucle à l'aide de GOTO

L'instruction GOTO s'avère précieuse lorsque l'on veut confier à l'ordinateur une tâche qui se répète indéfiniment, ou lorsque, en d'autres circonstances, l'on veut changer l'ordre d'exécution des lignes d'un programme. GOTO peut être utilisé pour la réalisation d'une boucle demandant à l'ordinateur de revenir en arrière et de répéter l'exécution d'une instruction ou d'une série d'instructions. En voici quelques exemples :

Essayez :

```
┌→ 10 PRINT "*";
└─ 20 GOTO 10
```

La petite flèche, à gauche du programme, est simplement là pour vous donner une représentation graphique de la boucle. L'ordinateur imprime, les unes après les autres, des rangées d'astérisques. Nous espérons que vous n'avez pas omis de taper le point-virgule (sinon, l'ordinateur aurait imprimé une seule colonne à la gauche de l'écran). Rien ne lui indique de s'arrêter. Appuyez sur la touche BREAK quand vous en aurez assez !

Essayez :

```
┌→10  CLS
│  20  PRINT@ (8,10), "URGENCE"
└─ 30  GOTO 10
```

L'ordinateur efface l'écran et réimprime le message à chaque fois qu'il boucle un tour : c'est ce qui le fait clignoter.

Essayez :

```
┌→ 10  PRINT "/"
└─ 20  GOTO 10
```

Belle série de diagonales, on doit sûrement pouvoir l'exploiter !

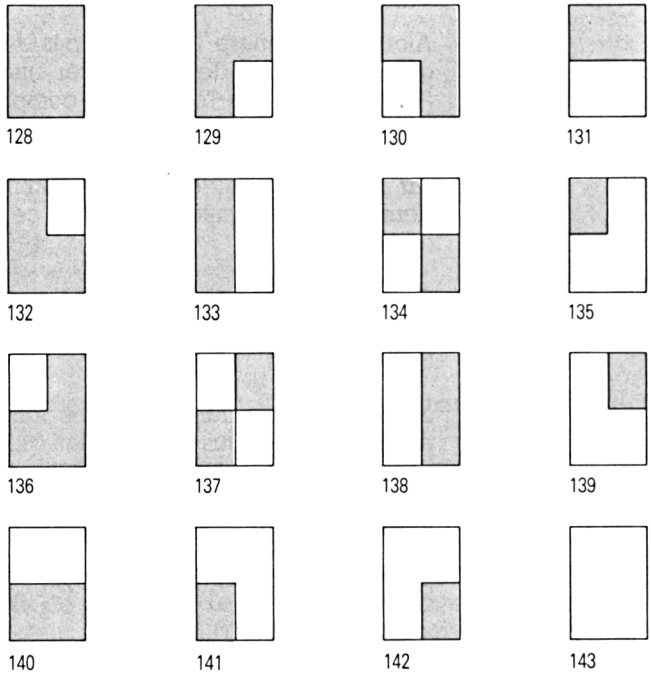
Tous ces programmes sont des illustrations de ce que l'on peut appeler des « boucles sans fin ». Dans les prochains chapitres, nous découvrirons comment décompter et comment arrêter les tours de boucle opérés par l'ordinateur.

Les caractères graphiques

Nous avons tracé beaucoup de petits dessins, parce que c'est une manière simple et récréative de s'initier au mode d'affichage et de se familiariser avec les instructions qui le contrôlent. Jusqu'à maintenant, nous nous sommes servis des lettres de l'alphabet et des chiffres. Nous allons apprendre à utiliser un nouvel « outil » que presque tous les micro-ordinateurs possèdent : les caractères graphiques.

Ils ne figurent pas à notre clavier, mais notre manuel nous en fournit la liste (voir figure). Un code spécial permet de les imprimer à l'écran (consultez votre clavier ou votre manuel).

Caractères graphiques
Voici l'inventaire de toutes les formes graphiques dont dispose notre ordinateur. Sous ces formes figurent les codes CHR$ qui permettent de les obtenir. Chaque symbole graphique occupe un carré de la grille d'écran, c'est-à-dire qu'il occupe le même espace qu'une lettre ou un nombre.

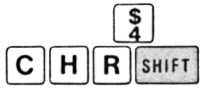


En réalité, à chaque caractère que l'ordinateur peut imprimer, lettres et nombres inclus, correspond un code numérique. Les touches de contrôle, telles ENTER, BREAK et les touches du curseur en possèdent un également. Chaque fois que nous appuyons sur une touche, le code caractère est transmis à l'ordinateur,

qui fait alors apparaître sur l'écran ce que nous voulons y voir affiché. La plupart des ordinateurs, y compris le nôtre, utilisent un système de codification fondé sur l'American Standard Code for Information Interchange ou ASCII (l'ASCII ne couvre pas les caractères graphiques : chaque fabricant d'ordinateurs utilise ses codes spécifiques). En dessous de chaque symbole graphique, à la page 51, se trouve un nombre : c'est le code qui permet de l'identifier.

Quand nous tapons un numéro de code, nous devons informer l'ordinateur qu'il ne s'agit pas d'un nombre ordinaire. Si nous rédigeons l'inscription de cette manière :

```
PRINT 137
```



Alors, il imprimera 137 et non pas deux petits carrés en diagonale. Nous devons utiliser une fonction spéciale (CHR$), pour que l'ordinateur comprenne. Vous souvenez-vous des fonctions du chapitre précédent ? CHR$ demande à l'ordinateur d'effectuer une manipulation sur le nombre qui le suit, tout comme SQR ou TAN ; il demande à l'ordinateur de traiter ce nombre comme un code ASCII.

Essayez :

```
PRINT CHR$(137)
```

Nous pouvons additionner des fonctions CHR$ entre elles, comme nous l'avions fait pour les chaînes au chapitre dernier.

Essayez :

```
PRINT CHR$(136)+CHR$(128)+CHR(128)
```

Un camion

Ce qui donne un camion schématisé, comme nous le montre l'illustration.

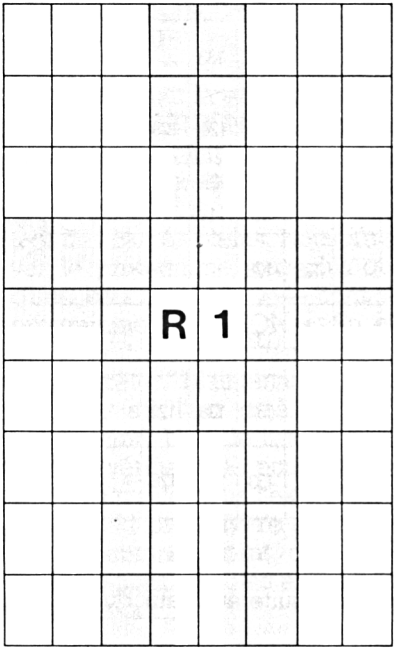
Maintenant que nous connaissons quelques codes CHR$, nous pouvons les utiliser pour dessiner des images.

Dessin d'un robot

Esquissons la silhouette d'un robot, que nous rendrons plus vivant en lui plaçant un message clignotant sur la poitrine. Prenons d'abord un bloc de papier quadrillé pour y dessiner la silhouette (voir la figure). Nous avons

choisi un ensemble de motifs très simple dans notre assortiment de caractères graphiques.

Silhouette du robot
Voici notre robot. Nous dessinons la silhouette en plaçant des caractères graphiques sur les cases de notre tableau-écran. Sur la poitrine du robot apparaîtra le message clignotant "R1".



Passons maintenant à la programmation. Nous prenons une autre feuille de papier quadrillé et nous notons, case par case, le code des caractères que nous allons utiliser, ainsi que les coordonnées du robot (rangées, colonnes). Nous rédigeons ensuite le programme sur papier. Vous pouvez le taper directement sur l'écran si votre ordinateur offre de bonnes possibilités de listage et d'édition (vous permettant de vérifier et de modifier des lignes de programme que vous avez déjà tapées). Si nous écrivions un programme très long, nous devrions le tester section par section. Mais comme celui-ci est assez court, nous le rédigerons d'une traite, puis nous l'exécuterons.

Commençons par les instructions préliminaires :

```
10 REM *** LE ROBOT CLIGNOTANT ***
20 REM PAR SUSAN CURRAN
30 CLS
```

Plan du robot

Sur ce second dessin, nous avons remplacé les caractères graphiques par leurs numéros de code CHR$ correspondant. La partie clignotante du robot fera alterner deux caractères graphiques et deux caractères alphanumériques.

			128	128			
			128	128			
			138	133			
128	128	128	128	128	128	128	128
131	131	128	128/ R	128/ 1	128	131	131
		128	128	128	128		
		128	133	138	128		
		128	133	138	128		
		128	133	138	128		

Passons ensuite au cœur du programme :

```
100 REM DESSIN D'UN ROBOT
110 PRINT@ (5,15), CHR$(128)+CHR$(128)
120 PRINT@ (6,15), CHR$(128)+CHR$(128)
130 PRINT@ (7,15), CHR$(138)+CHR$(133)
140 PRINT@ (8,12), CHR$(128)+CHR$(128)+CHR$(128)
    +CHR$(128)
145 PRINT@ (8,16), CHR$(128)+CHR$(128)+CHR$(128)
    +CHR$(128)
150 PRINT@ (9,12), CHR$(131)+CHR$(131)+CHR$(128)
    +CHR$(128)
155 PRINT@ (9,16), CHR$(128)+CHR$(128)+CHR$(131)
    +CHR$(131)
160 PRINT@ (10,14), CHR$(128)+CHR$(128)+CHR$(128)
    +CHR$(128)
170 PRINT@ (11,14), CHR$(128)+CHR$(133)+CHR$(138)
    +CHR$(128)
180 PRINT@ (12,14), CHR$(128)+CHR$(133)+CHR$(138)
    +CHR$(128)
190 PRINT@ (13,14), CHR$(128)+CHR$(133)+CHR$(138)
    +CHR$(128)
```

Maintenant, la partie clignotante :

```
200 REM POITRINE CLIGNOTANTE
210 PRINT@ (9,15), "R1";
220 PRINT@ (9,15), CHR$(128)+CHR$(128);
230 GOTO 210
```

Nous avons fait là exactement la même chose que dans l'exemple « URGENCE » vu précédemment. Cette fois cependant, au lieu d'effacer l'écran pour faire clignoter le message, nous avons alterné l'affichage de « R1 » (pour ROBOT 1) avec celui des caractères représentant des carrés pleins. Nous n'avons pas besoin d'achever le programme, pas plus que nous n'avons besoin d'une boucle « vide » — comme c'était le cas pour le programme « MAISON ». En effet, grâce à la boucle de la poitrine clignotante, notre programme ne peut pas s'arrêter. Maintenant, mettez en marche l'ordinateur ou, s'il l'est déjà, tapez NEW, puis entrez le programme. Nous le listons, par groupes de dix lignes, afin de vérifier que nous avons correctement tapé les codes. Ensuite nous l'exécutons (RUN), et si le résultat ne répond pas à notre attente, alors nous le modifions et l'exécutons autant de fois qu'il sera nécessaire. Lorsque nous sommes enfin satisfait, nous demandons à l'ordinateur :

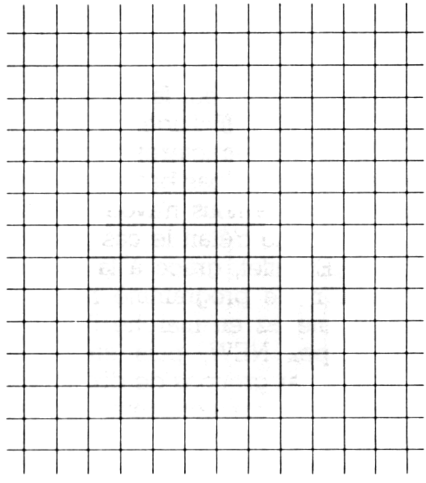
```
SAVE "ROBOT"
```

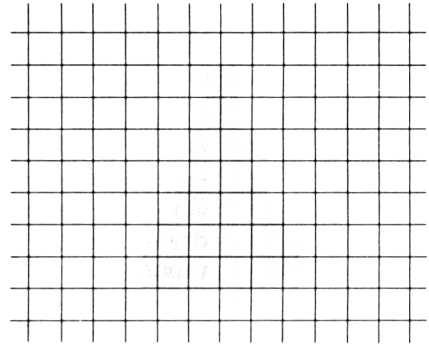
et nous appliquons la procédure de sauvegarde des programmes.

Maintenant, c'est à vous

Une fois que vous aurez exécuté le programme « ROBOT » avec succès, essayez d'écrire quelques programmes similaires. Ne reculez pas devant ce petit effort supplémentaire, et expliquez, à l'aide de la commande REM ce que vous allez faire. Ensuite, vous pouvez sauvegarder vos programmes sur cassettes. Nous ne vous donnerons pas de programmes tout faits.

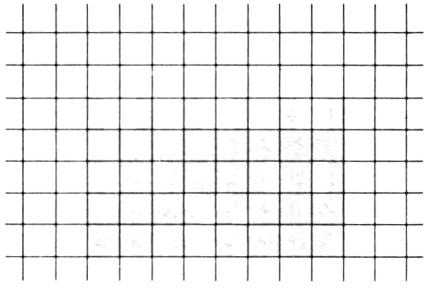
Nous vous suggérerons simplement quelques réalisations possibles. Déterminez vous-mêmes les caractères graphiques qu'il vous faudra employer ainsi que la meilleure façon de les positionner sur l'écran.

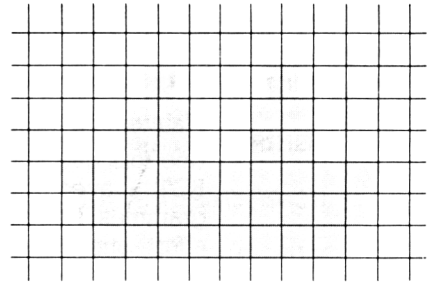




Silhouettes à réaliser sur votre écran

Faites dessiner à votre ordinateur la silhouette de la tour présentée ci-dessus, et de celle du chat à sa gauche. Vous devrez sûrement improviser pour obtenir les moustaches. Nous avons également esquissé un château et un navire de guerre : essayez de les réaliser.





Contrôle

Arrivé à la fin de ce chapitre, vous devriez savoir :

Sur la structure du programme

— comment les programmes se distinguent des instructions utilisées en mode direct ;
— comment utiliser les numéros de ligne ;
— quels sont les numéros de ligne que l'on utilise ;
— dans quel ordre les lignes peuvent être entrées ;
— comment s'utilisent, dans les programmes, les commandes suivantes :
  NEW
  LIST (numéro de ligne)
  LIST (numéro de ligne) — (numéro de ligne)
  LIST (numéro de ligne) —
  LIST — (numéro de ligne)
  RUN
  SAVE
— comment utiliser les mots clés suivants :
  END
  LOAD

Sur l'instruction REM

— comment on l'utilise et ce dont on peut la faire suivre ;
— l'endroit où l'on doit la placer sur une ligne contenant plusieurs instructions ;
— comment l'utiliser pour clarifier la structure d'un programme ;
— comment l'utiliser pour intituler un programme ;

Sur les noms attribués aux programmes

— quelle est la longueur maximale acceptée par votre ordinateur pour la sauvegarde des programmes ;

Sur la commande BREAK, STOP ou assimilée

— comment l'utiliser pour interrompre un programme pris dans une boucle sans fin ;

Sur les caractères graphiques

— quels sont les caractères graphiques disponibles sur votre ordinateur et sur le nôtre ;
— quels sont leurs codes CHR$;
— comment utiliser le code CHR$ dans les instructions d'affichage (PRINT).

3

Les variables

Au cours de ce chapitre, nous étudierons :
— les variables de chaînes et les variables numériques,
— l'instruction LET...=,
— l'instruction INPUT,
— les fonctions RND et INT.

La notion de variable est à la base de toutes les méthodes que le Basic utilise pour réaliser un maximum d'opérations en un minimum de lignes. Ce n'est pas un concept facile à maîtriser ; nous l'étudierons donc en profondeur dans ce chapitre.

Qu'est-ce qu'une variable ? Au chapitre 1, nous vous avons demandé d'essayer :

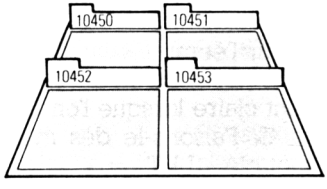
```
PRINT A
```

et l'ordinateur vous a donné 0. « A », utilisé de cette façon, est une variable. C'est le nom attribué à un emplacement dans la mémoire de l'ordinateur ; et comme nous n'y avons pas encore placé d'information, son contenu est égal à 0.

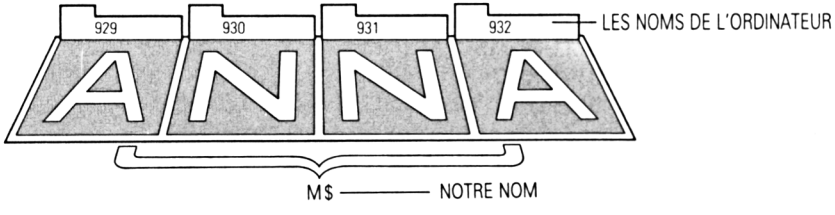
Entrons un peu plus dans les détails. Vous savez certainement que l'ordinateur dispose d'un espace mémoire, qui est occupé en partie par les programmes résidents. Néanmoins, la plus grande part de cet espace est disponible pour écrire des programmes (ce type de mémoire est appelé mémoire vive ou MEV. Les informations qu'elle contient peuvent être organisées, modifiées ou supprimées selon notre bon vouloir et celui de l'ordinateur). La machine utilise sa mémoire pour stocker toutes les informations que vous lui communiquez quand vous écrivez un programme : instructions, nombres, chaînes de données, numéros de ligne, etc., ainsi que toutes les informations qui lui sont nécessaires pour les manipulations ou les calculs qu'il effectue.

Quand vous programmerez en Basic, l'ordinateur décidera pour vous des emplacements où stocker les informations qui lui seront fournies. Chaque emplacement a un numéro d'adresse. Si votre ordinateur a une MEV de 16 Ko (c'est-à-dire si sa mémoire vive contient 16 384 emplacements différents), alors, les numéros iront de 0 à 16 383.

Cependant, il est souvent intéressant de repérer une information qui se trouve en mémoire. Pour ce faire, nous avons besoin de connaître son adresse. L'ordinateur nous laisse choisir un nom dont nous nous servirons pour repérer un emplacement mémoire ou une série d'emplacements (chaque emplacement ne peut contenir qu'une lettre ou son équivalent, aussi faut-il avoir recours à un groupe d'emplacements pour mémoriser une chaîne, par exemple). « A » est un exemple de nom que nous pouvons choisir pour une variable. Certains ordinateurs (pas le nôtre) autorisent des noms beaucoup plus longs. Nous en reparlerons un peu plus loin.



Les emplacements mémoire
L'ordinateur stocke l'information dans des emplacements numérotés et nous laisse choisir leurs noms. Grâce à eux, nous pourrons ranger ou sortir des données qui se trouvent dans les emplacements.



Pour attribuer un nom à un emplacement, il suffit de le mentionner dans une ligne de programme. Ainsi, lorsque nous entrons une ligne telle que :

```
PRINT C
```

l'ordinateur réagit en réservant un emplacement mémoire, ou une série d'emplacements (cela dépend du type de nom attribué), qu'il nous laissera baptiser C. A chaque fois que nous mentionnerons C, il comprendra que nous sommes intéressés, non pas par la lettre C, mais par l'information qui se trouve à l'emplacement C. Alors, lorsque nous lui demanderons d'imprimer C, il imprimera, non pas la lettre, mais l'information contenue dans C (la « donnée »).

En conséquence, lorsque l'on écrit un programme, il n'est pas nécessaire de savoir à l'avance quel sera le contenu de C. Ainsi, l'information peut être entrée par l'utilisateur au moment de la mise en route du programme, en réponse à une question du type : « Quel est le montant du dernier chèque que vous avez débité de votre compte en banque ? » (Nous allons voir d'ici peu comment cela peut se faire.) C peut également contenir le résultat d'un calcul effectué par l'ordinateur, gardé en mémoire pour une utilisation ultérieure. Ou encore, C peut contenir des lots successifs de valeurs différentes, qui sont tous utilisés dans le même but :
— pour les multiplier par une certaine somme,
— pour les imprimer sur une liste,
— pour indiquer l'affichage sur l'écran.

La distinction entre variable (C) et donnée (l'information contenue dans C) devient claire lorsque l'on donne effectivement un contenu à C. Faisons-le dès maintenant, grâce à un nouveau mot clé « LET...= ».

Essayez :

```
10 REM PROGRAMME SUR LES VARIABLES
20 LET C=15
30 PRINT C
40 END
```

Quelle est la sortie produite par l'ordinateur ?

```
15
```

bien sûr !

LET signale à l'ordinateur que nous glissons une donnée dans une variable (c'est-à-dire dans un emplacement de la MEV). Il existe d'autres façons d'associer données et variables, mais nous en resterons là pour le moment. On utilise la fonction « = » après LET pour

indiquer la donnée à l'ordinateur. Notez que « = » utilisé de cette manière ne correspond pas au symbole arithmétique que vous connaissez; vous ne pouvez retourner l'instruction et dire :

```
LET 15=C
```

Vous devez d'abord indiquer l'emplacement, et ensuite seulement le contenu. On peut considérer que le symbole « = » signifie « contient » plutôt que « est égal à », ainsi « LET C=15 » veut dire « l'emplacement C contient la valeur 15 ».

On peut changer le contenu de la variable C pourvu que l'on en informe l'ordinateur en écrivant LET C= quelque chose d'autre.

Essayez :

```
10 REM CHANGEMENT DU CONTENU D'UNE VARIABLE
20 LET C=15
30 PRINT C
32 LET C=25
33 PRINT C
40 END
```

Si vous n'avez pas effacé le programme précédent vous n'avez pas besoin de réécrire tout celui-ci. Ajoutez simplement les lignes 32 et 33. Quelle est la sortie produite par l'ordinateur ? Voici :

```
15
25
```

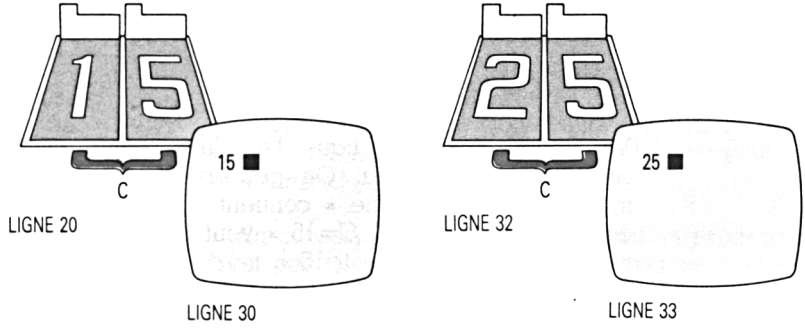
C reste le nom de l'emplacement, son contenu devient successivement 15 puis 25, et deviendra ce que l'on y mettra par la suite.

Nous pouvons obtenir le même résultat en associant à C une somme arithmétique. Les lignes 32 et 33 deviennent alors :

```
32 LET C=12+13
33 PRINT C
```

L'ordinateur effectue l'addition pour nous et imprime 25.

Utilisation de LET



LET ne permet pas seulement d'affecter des données à un emplacement mémoire, il offre également la possibilité d'en changer le contenu. Dans le programme de la page 61, la ligne 20 affecte le nombre 15 à l'emplacement C et la ligne 30 imprime le contenu de C. La ligne 32 change ce contenu, et la ligne 33 imprime le nouveau contenu, 25.

Nous pouvons même lui demander de faire un peu de calcul avec le contenu originel ; remplaçons la ligne 32 et 33 par :

```
32 LET C=C+10
33 PRINT C
```

Voilà qui peut sembler paradoxal, mais tout ce que nous avons fait, c'est ajouter dix à la valeur initiale de C (ce qui nous donne 25). Bien sûr, cette fois, notre instruction LET ressemble encore moins à une équation ; vous pouvez voir du premier coup d'œil que cette expression ne serait pas recevable en algèbre, car aucune valeur de C ne pourrait établir l'égalité. En changeant son contenu, nous pouvons ainsi utiliser C dans une boucle du même type que celles que nous avons formées au chapitre précédent.

Essayez :

```
 10 REM BOUCLE DE VARIABLE
 20 LET C=1
→30 PRINT C
 40 LET C=C+1
 50 GOTO 30
```

Cette fois, l'instruction LET ajoute simplement 1 au contenu de C, à chaque tour de boucle.

Voici ce que nous obtenons :

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

Appuyez sur BREAK quand vous en avez assez.

Les noms de variables

Les noms de variables sont différents selon qu'il s'agit d'une chaîne ou d'un nombre. Si l'emplacement C est affecté à une variable numérique, l'on ne peut plus permuter son contenu avec une chaîne. Il faut que le nouveau contenu soit un nombre. Voici les noms que notre ordinateur accepte :

Pour les variables numériques

Noms formés d'un ou deux caractères, dont le premier doit être une lettre majuscule et le second, une lettre majuscule ou un nombre. Quelques exemples de noms recevables :

```
AA
A1
Z9
```

En voilà d'autres qui ne le seront pas :

2A (Le nombre précède la lettre : ce doit être le contraire)
ABC (trop long)
A? (Les symboles autres que les chiffres et les lettres ne sont pas acceptés).

Pour les variables de chaîne

Ce sont les mêmes noms que ceux réservés aux nombres, mais, cette fois, suivis du signe $. Quelques exemples de noms recevables :

```
A$
W4$
```

En voici d'autres qui ne le seront pas :

```
B    (absence de signe $)
4C$  (le nombre précède la lettre)
X?   (? n'est pas le signe qui convient, ce devrait être $)
```

Si l'on tente de placer une chaîne dans un emplacement désigné par une variable numérique, l'ordinateur réagira par un message d'erreur. On peut toujours placer un nombre dans le « tiroir » d'une variable de chaîne, puisque les nombres peuvent figurer sans restriction dans une chaîne, mais vous ne pouvez vous livrer à des opérations de calcul sur cette variable, comme ce serait le cas avec une variable numérique, et vous ne devez pas omettre les guillemets au début et à la fin de la chaîne !

Les variables numériques

Tant que l'on travaille avec des variables numériques, il est possible de les employer dans des calculs, comme si c'étaient des nombres ; bien sûr, l'ordinateur, lui, manipule les nombres contenus dans les emplacements et non pas les noms de variables.

Essayez :

```
10 REM VARIABLES NUMERIQUES
20 LET X = 2 : LET Y = 3
30 PRINT X + Y
40 PRINT X * Y
50 END
```

Voici un exemple qui illustre de façon plus parlante les calculs qui peuvent s'effectuer sur les variables.

Essayez :

```
10 REM VARIABLES NUMERIQUES II
20 LET X = 5 : LET Y = 6
30 PRINT X;"+";Y;"=";X + Y
40 PRINT X;"*";Y;"=";X * Y
50 PRINT X;"/";Y;"=";X / Y
60 PRINT X;"-";Y;"=";X - Y
70 PRINT"RACINE CARREE DE ";X;"=";SQR(X)
80 END
```

Vous pouvez également utiliser des variables pour les coordonnées d'affichage. Voici un autre exemple de

programme de « remplissage de l'écran », mais avec une particularité : il fonctionne à l'envers !

Essayez :

```
10 REM CHARMEUR DE SERPENT
20 CLS
30 LET N = 15
40 PRINT @ (N,10), "SSSSS";
50 LET N = N - 1
60 GOTO 40
```

Vous obtiendrez un message d'erreur quand N deviendra négatif !

Les variables de chaîne

Abordons maintenant les variables de chaîne. Elles sont très utiles pour différentes raisons. Tout d'abord, elles réduisent le temps de frappe.

Essayez :

```
10 MOTIF
15 CLS
20 LET A$ = "** /** /"
30 LET B$ = "--------"
40 PRINT A$;A$;A$;A$;
50 PRINT B$;B$;B$;B$;
60 GOTO 40
```

Ce programme est conçu pour un écran de 32 colonnes ; aussi, à chaque fois que les lignes 30 et 50 sont imprimées, chacune d'elle remplit une ligne de l'écran. Faites les modifications nécessaires si la taille de votre écran est différente.

Essayez :

```
10 REM L'ÉCRAN BAVARD
15 CLS
20 LET A$ = "DES MOTS, TOUJOURS DES MOTS"
30 PRINT A$;
40 GOTO 30
```

Nous pouvons également associer des codes de caractères graphiques à des variables de chaîne, en utilisant des instructions telles que :

```
LET A$ = CHR$(128)+CHR$(128)+CHR$(128)
```

Ce qui nous facilite le travail pour les dessins (comme celui du robot au chapitre précédent). Bien sûr, il est possible d'ajouter des variables de chaîne, comme ceci :

```
10 LET A$ = "AUTO"
20 LET B$ = "ROUTE"
30 LET C$ = A$ + B$
40 PRINT C$
```

Il est également possible d'ajouter des chaînes à des variables de chaîne :

```
10 LET A$ = "AVANT"
20 LET B$ = A$ + "AGE"
30 PRINT B$
```

On peut, enfin, modifier le contenu d'une variable de chaîne en lui ajoutant une autre chaîne :

Essayez :

```
 10 REM CHAINE CROISSANTE
 20 LET A$ = "*"
 30 PRINT A$
 40 LET A$ = A$ + "*"
 50 GOTO 30
```

Pas de point-virgule à la fin de l'instruction PRINT, cette fois, contrairement au programme « l'écran bavard », car nous voulons voir la ligne d'astérisques croître régulièrement, en forme de triangle, comme ceci :

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

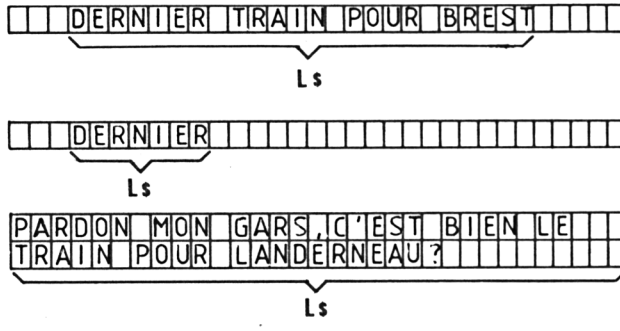
Si vous laissez le programme s'exécuter suffisamment longtemps, vous verrez apparaître un message d'erreur : lorsque A$ deviendra trop long. De la même manière que nous pouvons allonger A$, il nous est toujours possible de le raccourcir, comme dans ce programme :

Essayez :

```
10 REM VARIABLE DE CHAINE RACCOURCIE
20 LET A$ = "DERNIER TRAIN POUR BREST"
30 PRINT A$
40 LET A$ = "DERNIER"
50 PRINT A$
60 END
```

L'ordinateur vous donnera :

```
DERNIER TRAIN POUR BREST
DERNIER
```

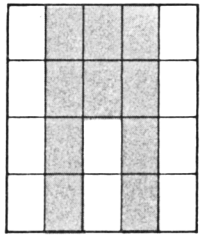


Changer le contenu d'une variable
Si nous associons une chaîne telle que « DERNIER TRAIN POUR BREST » à une variable de chaîne appelée L$, nous pouvons demander à l'ordinateur de raccourcir L$, comme dans le programme ci-dessus. Nous aurions pu également rallonger L$. Dans les deux cas, l'ordinateur change automatiquement la taille de L$.

Comme vous le voyez, l'ordinateur a effacé toute l'information contenue dans A$, quand nous lui avons demandé d'y placer autre chose. Il ne s'est pas contenté de libérer suffisamment d'espace pour pouvoir introduire la nouvelle chaîne.

Maintenant, c'est à vous

1) Laquelle de ces instructions notre ordinateur acceptera-t-il, et lesquelles d'entre elles provoqueront l'affichage d'un message d'erreur ?



a) LET A = "POMMES"

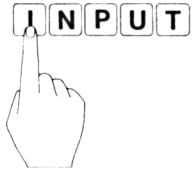
b) LET BC = 99

c) LET X$ = 56

2) Dessinez cette figure en fabriquant des chaînes qui contiennent les codes de caractères graphiques nécessaires ;

3) Placez votre nom dans une variable de chaîne puis demandez à l'ordinateur de l'afficher sur la ligne du haut, sur la ligne du milieu et enfin, sur celle du bas.

L'instruction INPUT



Étudions maintenant une autre façon d'associer des données à des variables grâce au mot clé INPUT. Il permet d'entrer une information dans le programme pendant qu'il tourne. Vous tapez l'information de la même manière que lorsque vous entrez un programme ou travaillez en mode direct. Mais cette fois, le programme utilise l'information sur le champ. Il doit cependant stocker la donnée quelque part aussitôt que vous l'avez entrée, même s'il lui faut l'utiliser à la ligne suivante. Pour identifier l'endroit où l'ordinateur place l'information, vous lui donnez, vous l'avez deviné, un nom de variable.

Voici comment mettre en œuvre l'instruction INPUT :

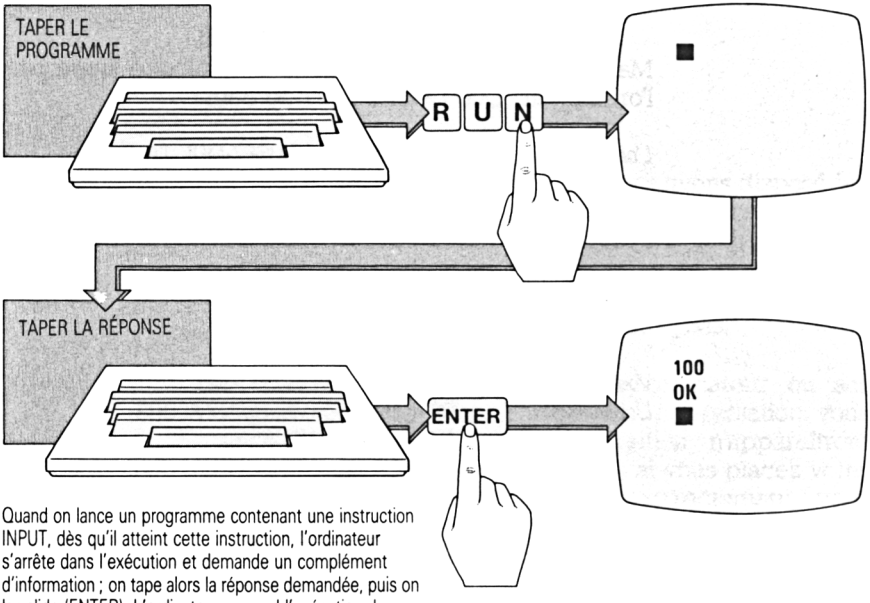
Essayez :

```
10  REM PROGRAMME INPUT
20  PRINT "QUEL EST VOTRE NOM?"
30  INPUT N$
40  PRINT "SALUT,  ";N$
50  END
```

Que se passe-t-il quand vous mettez le programme en route ? L'ordinateur imprime le message de la ligne 20 et le fait suivre d'un point d'interrogation (enfin, c'est ce que fait le nôtre) pour vous signifier qu'il attend que vous entriez une information. Il n'ira pas plus loin tant

que vous ne l'aurez pas fait ; alors, tapez votre nom, ou quelque chose de drôle, et enfoncez la touche ENTER pour dire que vous avez terminé. L'ordinateur associe votre nom à la variable N$ (une variable de chaîne, bien sûr) et, ensuite, l'imprime sur la ligne suivante.

Pour entrer une information



Quand on lance un programme contenant une instruction INPUT, dès qu'il atteint cette instruction, l'ordinateur s'arrête dans l'exécution et demande un complément d'information ; on tape alors la réponse demandée, puis on la valide (ENTER). L'ordinateur reprend l'exécution du programme en exploitant la nouvelle information.

L'exécution du programme se présente ainsi :

```
QUEL EST VOTRE NOM?
?PAUL MARTIN
SALUT, PAUL MARTIN
OK
```

Vous pouvez recourir à cette technique pour engager une conversation simple avec votre ordinateur. Essayons d'en programmer une au cours de laquelle l'ordinateur nous demandera :

— notre nom,
— le temps qu'il fait,
— si nous sommes en bonne forme.

Nous appellerons ces trois éléments d'information N$, T$ et F$. Cette notation nous aidera à nous souvenir de ce que recouvre chaque variable. Nous pouvons donc élaborer notre programme en établissant, tout d'abord, la liste des variables.

Variables :
Nom : N$
Temps : T$
Forme : F$

Maintenant imaginons la partie du dialogue jouée par l'ordinateur. Que pensez-vous de ceci ?

L'ordinateur : REBONJOUR, EXCUSEZ-MOI, J'AI OUBLIÉ VOTRE NOM - - -
Vous : RUFUS.
L'ordinateur : BIEN SÛR ! CONTENT DE VOUS REVOIR, RUFUS, FAIT-IL BEAU ?
Vous : IL PLEUT.
L'ordinateur : EH OUI, IL PLEUT - - - COMMENT ALLEZ-VOUS ?
Vous : JE SUIS EN PLEINE FORME.
L'ordinateur : EN PLEINE FORME, HEIN ? ALORS AU TRAVAIL, RUFUS. DONNEZ-MOI VOS INSTRUCTIONS. OK

Ceci est loin d'être parfait, car bon nombre des réponses que nous pourrions donner rendent les répliques de l'ordinateur quelque peu saugrenues, mais ce n'est là qu'un début. Passons maintenant à la programmation. Premièrement, la présentation :

```
10  REM *** RETROUVAILLES ***
20  CLS
```

ensuite la conversation proprement dite :

```
30  PRINT "REBONJOUR, EXCUSEZ-MOI, J'AI
    OUBLIE VOTRE NOM...."
40  INPUT N$
50  PRINT "BIEN-SUR ! CONTENT DE VOUS
    REVOIR, ";N$;". FAIT-IL BEAU "
```

```
60  INPUT T$
70  PRINT "EH OUI, ";T$;" ..."
80  PRINT "COMMENT ALLEZ-VOUS"
90  INPUT F$
100 PRINT F$;" EN PLEINE FORME, HEIN ?
    ALORS AU TRAVAIL, ";N$
110 PRINT "DONNEZ-MOI VOS INSTRUCTIONS"
```

et, pour terminer :

```
120 END
```

Avez-vous observé la façon dont nous avons disposé les guillemets ainsi que les espaces que nous y avons introduits, afin d'obtenir une sortie plus élégante ? Si la présentation de votre sortie ne vous paraît pas satisfaisante, rajoutez des espaces de telle manière que les mots ne soient pas coupés en fin de lignes.

Essayez votre programme sur ordinateur ou sur papier et voyez à quels types de conversation vous aboutissez. Les points d'interrogation n'apparaîtront peut-être pas au bon endroit, mais si vous placez votre demande d'information (INPUT) correctement, vous devriez obtenir un résultat relativement satisfaisant.

Pour entrer deux données ou plus

Une seule instruction INPUT suffit à entrer des données qui se répartiront dans plusieurs emplacements de variable. Vous devez énumérer tous les noms de ces variables, comme suit :

```
INPUT A, B, C,
```

Ensuite, l'ordinateur attendra que vous ayez entré chaque donnée. Sur notre ordinateur, nous répondrons sur une seule ligne, en séparant nos réponses par des virgules.

Abordons un exemple qui nous montre comment entrer (INPUT) deux données à la fois en utilisant des variables numériques.

Essayez :

```
10 REM *** CALCULS ***
20 PRINT "JE VAIS MULTIPLIER DEUX
   NOMBRES"
30 PRINT "QUE VOUS ALLEZ CHOISIR"
40 PRINT "ET JE VOUS DONNERAI LE
   RESULTAT"
50 PRINT "ENTREZ VOS NOMBRES"
60 INPUT X,Y
70 PRINT X;" FOIS ";Y;" DONNE ";X*Y
80 END
```

Notez bien l'usage que nous avons fait des instructions PRINT pour que l'ordinateur nous décrive ce qu'il fait. C'est particulièrement important dans un programme qui nous demande d'entrer des données, car sinon, l'on court le risque de se retrouver face à des points d'interrogation sans savoir ce que veut l'ordinateur. Nous avons utilisé un grand nombre de courtes instructions PRINT, plutôt qu'un petit nombre d'instructions longues. Ceci, afin qu'aucun des mots ne soit interrompu en bout de ligne : il est plus commode de procéder ainsi plutôt que d'essayer de prévoir les endroits où il faudra placer des blancs.

Il est tellement important de savoir ce que l'ordinateur va faire et ce qu'il nous demande, que sur le nôtre, il est possible de combiner l'instruction INPUT avec l'impression d'une chaîne de caractères. Voici comment nous procédons :

```
10 INPUT "DONNEZ-MOI UN NOMBRE ";N
20 PRINT "VOTRE NOMBRE EST ";N
30 INPUT "DEUX NOMBRES CETTE FOIS ";M,P
40 PRINT "VOUS AVEZ CHOISI ";M;" ET ";P
50 INPUT "ET MAINTENANT UN MOT OU UN
   GROUPE DE MOTS ";W$
60 PRINT "CETTE FOIS, VOUS M'AVEZ DONNE
   ";W$
70 PRINT "VOTRE LISTE, AU COMPLET : ",N
   ,,,,M,,,,P,,,,W$
80 END
```

Avez-vous bien compris ce que nous venons de faire ? Pour en être sûr, essayez le programme et établissez vous-même la liste des variables. Examinez avec soin la ponctuation ; avez-vous remarqué que nous avons laissé un espace à la fin de la chaîne figurant ligne 60 (elle est, en effet suivie d'une variable de chaîne), mais que nous ne l'avons pas fait pour les chaînes suivies d'une

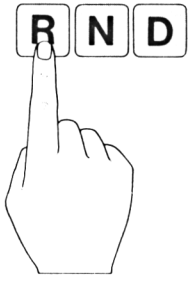
variable numérique, puisque l'ordinateur laisse automatiquement un espace dans ce cas? Notez également l'emploi des virgules, pour afficher les variables sur une colonne (ligne 70).

Maintenant, c'est à vous

4) Écrivez un programme qui amène l'ordinateur à vous réclamer deux nombres X et Y, puis à vous en donner la somme (X + Y), la différence (X − Y), le produit (X * Y), et le quotient (X / Y).

5) Écrivez un programme où l'ordinateur vous demande d'entrer une première chaîne de caractères graphiques qu'il imprimera, puis une seconde, qu'il imprimera également.

Les nombres aléatoires



Pour clore ce chapitre, nous allons étudier la fonction « nombre aléatoire » RND qui permet également de modifier les informations.

Il en existe de nombreuses versions. Vérifiez dans le manuel d'utilisateur que votre ordinateur accepte la nôtre.

L'ordinateur ne peut pas lancer les dés! Mais il peut profiter de l'énorme complexité de ses circuits pour sortir un nombre qui soit aussi aléatoire que celui obtenu par un jet de dé. Notre ordinateur produit des nombres aléatoires compris entre 0 et 1, aussi sont-ils décimaux, avec neuf chiffres après la virgule. Voyons ce à quoi ils ressemblent.

Choisir des nombres au hasard
La roulette permet de produire des nombres totalement au hasard. L'ordinateur offre également cette possibilité, grâce à la fonction RND.

Essayez :

```
10 PRINT RND
20 GOTO 10
```

Arrêtez l'ordinateur après qu'une page d'écran ou deux auront été remplies, afin de pouvoir jeter un coup d'œil à ces nombres. Ils se présentent ainsi :

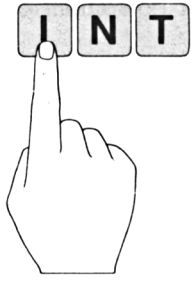
```
.753755291
.984257657
.528055965
.462307892
.333420012
```

Remarquez que RND est une fonction : elle n'est pas utilisée en tête d'une instruction. C'est le mot clé PRINT qui indique à l'ordinateur ce qu'il doit faire du résultat de l'expression RND.

Les nombres aléatoires utilisés dans un programme simulant un lancer de dés doivent être des nombres entiers. Comment faire pour transformer des nombres décimaux en nombres entiers. Il faut procéder par étapes :

a) multiplier les nombres aléatoires par un entier, pour définir la grandeur des nombres désirés. Par exemple, multiplier par 6, comme les 6 faces du dé, nous permettra d'obtenir des nombres compris entre 0 et 6 :

b) transformer ces nombres en entiers grâce à la fonction INT (abréviation de integer, nombre entier en anglais). INT nous donne l'entier immédiatement inférieur au nombre décimal, en d'autres termes, il l'arrondit. Ainsi :




```
INT (5,5)
```

donne 5 ; et

```
INT (79,23)
```

donne 79.

La combinaison des étapes a) et b) nous donne ce type d'instruction courte :

```
PRINT INT (RND*6)
```

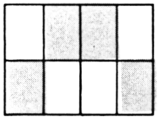
Les parenthèses indiquent à l'ordinateur de faire d'abord la multiplication avant d'arrondir le nombre décimal obtenu. INT (RND)*6 nous donnerait l'entier inférieur aux nombres aléatoires compris entre 0 et 1 (c'est-à-dire 0, dans tous les cas), multiplié par 6 (à nouveau 0, nous serions bien avancés !).

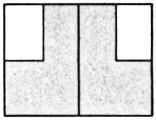
Il reste encore une chose à prendre en considération. Puisque la fonction INT arrondit les nombres, cette instruction devrait, en fait, nous donner une série de nombres aléatoires échelonnés de 0 à 5 et non pas de 1 à 6 comme avec un dé. (Essayez pour vérifier, si vous le désirez.) Aussi, pour obtenir des nombres entiers, aléatoires, compris entre 1 et 6, notre instruction définitive sera :

```
PRINT 1+INT(RND*6)
```

Bien sûr, l'on peut produire des nombres aléatoires de n'importe quelle grandeur. Par exemple, pour obtenir des nombres compris entre 20 et 30, voici l'instruction qu'il faut donner à l'ordinateur :

```
PRINT 20+INT(RND*11)
```


Lièvre


Tortue

Nous allons maintenant écrire un programme simple utilisant RND. C'est une course. Un lièvre et une tortue se poursuivront d'un bord à l'autre de l'écran. A chaque fois, nous les ferons avancer d'un nombre aléatoire de colonnes, compris entre 1 et 4. Vous pouvez engager les paris...

Puisque nous n'avons pas encore appris à sortir d'une boucle sans appuyer sur BREAK, nous obtiendrons la succession des déplacements à l'aide d'une boucle sans fin en utilisant GOTO. Ce qui veut dire que notre course se terminera de façon plus élégante : par un message d'erreur.

Comment allons-nous procéder ? Et bien, utilisons deux variables de chaîne, L$ et T$, pour les silhouettes de notre lièvre et de notre tortue (voir figure) ; leur petite taille facilite les déplacements.

```
100 LET L$ = CHR$(137) + CHR$(134)
110 LET T$ = CHR$(136) + CHR$(132)
```

Nous utilisons des nombres élevés pour les numéros de ligne, afin de pouvoir rajouter d'autres instructions, si nous le désirons. Où allons-nous placer le lièvre et la tortue ? Mettons-les sur les rangées 6 et 8 de notre écran. Nous ferons avancer les colonnes sur lesquelles ils se trouvent, comme s'ils couraient. Nous leur associerons donc des variables numériques. Soit L pour le lièvre et T pour la tortue. L'ordinateur ne confondra pas celles-ci avec L$ et T$ puisqu'il sait que L et T sont nécessairement associées à des nombres et L$ et T$ à des chaînes ou à des caractères graphiques. Ce qui nous donnera des instructions PRINT du type :

```
PRINT @ (6, L), L$
```

C'est compris ? Bon, fournissons quelques précisions à l'ordinateur au sujet de L et T. Bien qu'il confère automatiquement à ces variables la valeur initiale 0, il est logique d'insérer dans le programme une ligne qui

nous rappelle les valeurs initiales que nous leur avons affectées :

```
120 LET L = 0 : LET T = 0
```

Vous pouvez utiliser deux lignes si vous préférez. Maintenant, amorçons notre boucle, et utilisons pour ce faire des numéros de ligne distinctifs :

```
200 PRINT@ (6,L), L$
210 PRINT@ (8,T), T$
```

ainsi que deux instructions qui permettent d'augmenter la valeur de L et T par un calcul aléatoire.

```
220 LET L = L + 1 + INT(RND*4)
230 LET T = T + 1 + INT(RND*4)
```

L'ordinateur prendra en compte l'ensemble de L+1+INT(RND*4) et, aussi compliqué que cela puisse paraître, le résoudra comme une expression. Ensuite, la valeur résultant de ce calcul sera affectée à L. Ainsi nous pourrons revenir à la ligne 200 et le lièvre s'affichera un peu plus loin. Cependant, il nous faut supprimer le lièvre précédent ou, sinon, nous obtiendrons une rangée de lièvres sur l'écran. Il y a plusieurs manières de parvenir à ce résultat. Nous choisirons la plus simple, qui nous permettra également de supprimer les tortues.

```
240 CLS
```

Cette instruction doit être, elle aussi, inclue dans la boucle, que nous pouvons maintenant fermer, avant de revenir en arrière pour faire progresser le lièvre et la tortue sur l'écran.

```
250 GOTO 200
```

Êtes-vous satisfait ? Réécrivons le programme en entier, ainsi que la liste des variables. Nous y ajouterons quelques éléments d'introduction :

Variables
Forme du lièvre : L$
Forme de la tortue : T$

Position de la colonne du lièvre : L
Position de la colonne de la tortue : T

```
10  REM *** LE LIEVRE ET LA TORTUE ***
20  REM          PAR SUSAN CURRAN
30  CLS
100 LET L$ = CHR$(137) + CHR$(134)
110 LET T$ = CHR$(136) + CHR$(132)
120 LET L = 0 : LET T = 0
190 REM DEBUT DE LA COURSE
200 PRINT@ (6,L), L$
210 PRINT@ (8,T), T$
220 LET L = L + 1 + INT(RND*4)
230 LET T = T + 1 + INT(RND*4)
240 CLS
250 GOTO 200
```

Pourquoi ne pas archiver ce programme, il se peut que vous désiriez le consulter à nouveau ?

Maintenant, c'est à vous

6) Écrivez un programme qui affiche, sur toute la surface de l'écran, des nombres aléatoires compris entre 17 et 26.
7) Écrivez un programme pour qu'un camion traverse l'écran, de gauche à droite, à une vitesse aléatoire déterminée avant le départ.

Contrôle

Arrivé à la fin de ce chapitre, vous devriez savoir :
Sur les variables
— ce qu'est un nom de variable ;
— quels noms votre ordinateur accepte pour les variables de chaîne et les variables numériques ;
— comment changer le contenu d'une variable ;
— comment utiliser l'instruction LET pour associer une donnée à une variable ;
— comment utiliser les noms de variables à la place de données dans des instructions PRINT, dans des expressions arithmétiques, etc.
Sur l'instruction INPUT
— comment utiliser INPUT pour que l'ordinateur interrompe l'exécution et attende qu'on lui communique une donnée ;

— comment attribuer un nom de variable à une entrée sollicitée par INPUT ;
— comment utiliser INPUT pour afficher une sollicitation à l'écran ;
— comment entrer plus d'une donnée à la fois.

Sur les fonctions RND et INT

— comment utiliser RND pour obtenir des nombres aléatoires compris entre 0 et 1 ;
— comment produire des nombres entiers aléatoires de grandeurs données ;
— comment utiliser généralement INT pour transformer les nombres décimaux en nombres entiers.

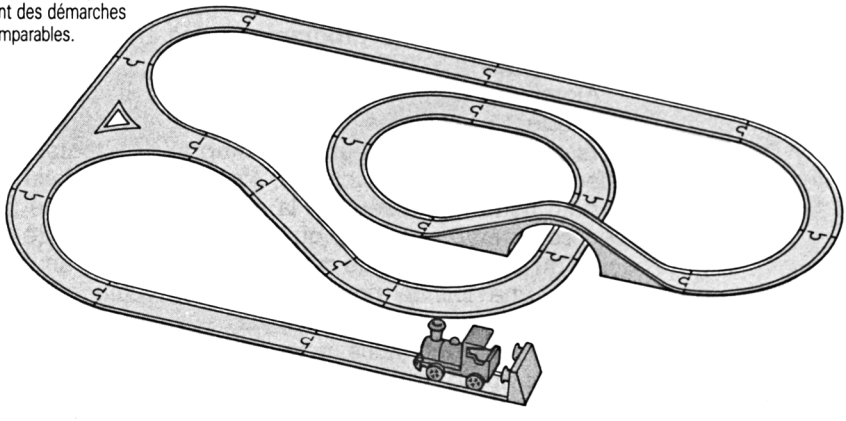
4

Boucles et branchements

Au cours de ce chapitre, nous étudierons :
— les boucles ouvertes et les boucles fermées,
— l'instruction IF... THEN,
— les instructions FOR... NEXT et STEP,
— les opérateurs logiques.

Notre jeune fils possède un train miniature. Le circuit se compose de segments de voie de longueurs diverses, d'embranchements, de ponts, qu'il faut assembler de telle façon que le train puisse se déplacer. C'est un travail qui n'est pas aussi facile qu'il y paraît, car vous devez concevoir votre circuit de manière à ce que les petites boucles se logent dans les grandes, que la courbe passe exactement sous le pont, etc.

Sur la bonne voie
Assembler correctement les segments de voie d'un circuit de train et s'assurer que les différentes parties d'un programme s'organisent logiquement sont des démarches comparables.



En un sens, élaborer un programme, c'est un peu réaliser l'assemblage d'un tel circuit. Les segments de voie sont comme les lignes du programme que vous devez disposer dans un ordre logique. De temps en temps, il y a un embranchement, un « nœud de décision » ; il faut choisir entre plusieurs itinéraires. Cela correspond à un aspect de la programmation que nous allons traiter, à savoir comment faire effectuer à l'ordinateur des tâches comportant des choix.

L'on trouve dans les programmes de grandes boucles dans lesquelles s'inscrivent parfois de plus petites, et l'on peut même trouver d'autres boucles à l'intérieur de ces dernières. Rien ne doit être négligé, afin d'éviter impérativement que certaines parties ne se raccordent à rien, ou ne débouchent sur rien. Il faut d'autre part s'efforcer de faire le maximum avec un minimum d'éléments. Si le travail est bien fait, alors le train pourra rouler sur ses rails ; de même, l'ordinateur pourra exécuter le programme, et cela avec efficacité, sans heurts.

Nous avons déjà examiné quelques-unes des instructions qui constituent les « voies » de votre programme. Dans ce chapitre, nous étudierons certaines des techniques utilisées pour mettre en place des boucles et des branchements.

Boucles fermées et boucles ouvertes

Nous avons déjà eu recours à l'instruction GOTO pour constituer des boucles fermées. Nous les appelons « boucles fermées » parce que lorsque l'ordinateur s'est engagé dans l'une d'entre elles, il n'a aucun moyen d'en ressortir ; à moins d'appuyer sur la touche BREAK. C'est un peu comme tirer la poignée de signal d'alarme d'un train. Aussi efficace qu'elle soit, cette méthode n'est pas idéale pour programmer des arrêts de sous-programmes !

Alors, comment ouvrir une boucle ? Il y a deux méthodes principales. Premièrement, nous pouvons demander à l'ordinateur de compter les tours qu'il effectue dans une boucle, et, lorsqu'il aura atteint un certain nombre, d'exécuter la suite du programme (un peu comme un aiguilleur qui compterait le nombre de tours qu'un train effectue sur une voie circulaire annexe, et qui, une fois le nombre de tours voulu effectué, l'aiguillerait sur la voie principale).

Deuxièmement, nous pouvons établir un branchement, un nœud de décision dans la boucle. À chaque fois que l'ordinateur effectuera un tour de boucle, nous

lui demanderons si telle condition est remplie — par exemple, si une variable que nous sommes en train d'incrémenter (dont nous augmentons régulièrement la valeur) a atteint la valeur désirée ou si le nombre entré à la suite d'une sollicitation (INPUT) est égal à un nombre donné. Si la réponse est « oui », et si c'est la réponse que nous attendons, nous aiguillons alors l'ordinateur sur une voie différente. Si c'est « non », nous le laissons entrer à nouveau dans la boucle.

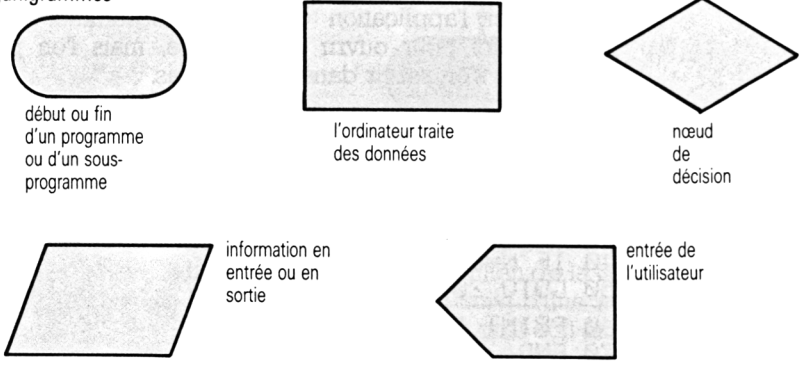
Nous étudierons d'abord une suite de commandes qui correspond à la deuxième méthode, et, un peu plus loin dans le chapitre, nous reviendrons à la première méthode.

Organigrammes

Vous avez peut-être déjà entendu parler des organigrammes. Ils sont, pour le programmeur, l'équivalent de notre circuit de train. Ce sont simplement des tableaux schématiques qui indiquent les itinéraires possibles ainsi que les critères sur lesquels l'ordinateur se fondera pour choisir l'un d'eux. Ils sont donc composés de voies (lignes fléchées indiquant la direction à suivre pour l'exécution du programme) jalonnées d'arrêts, où sont entreprises des actions (étapes où l'ordinateur exécute une commande particulière, par exemple l'impression d'une donnée), ponctuées d'aiguillages de décision (où le programme propose à l'ordinateur des choix), parsemées de jonctions où se raccordent de nouvelles entrées (INPUT), etc.

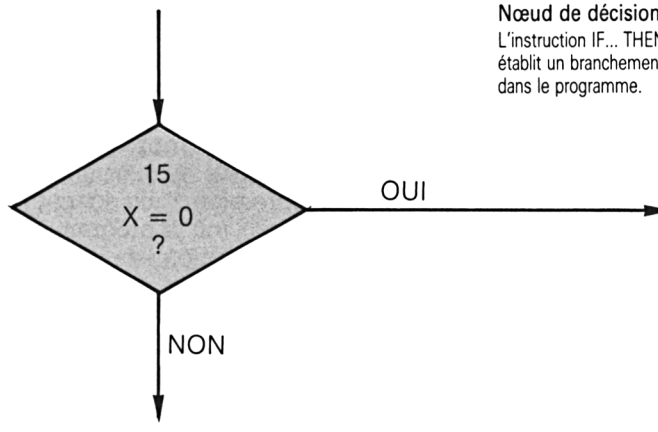
Un ensemble conventionnel de symboles est utilisé pour désigner les différents événements qui jalonnent l'organigramme. Cette figure vous présente ceux que nous utiliserons dans ce livre.

Symboles utilisés pour les organigrammes



Il vous revient de décider si vous utiliserez des organigrammes pour les programmes que vous écrirez vous-même par la suite. Il est certain que vous devrez planifier vos programmes, mais il vous paraîtra peut-être plus naturel d'avoir recours à des mots plutôt qu'à des symboles. Néanmoins, personnellement, nous apprécions les organigrammes ; nous en utiliserons un certain nombre dans ce livre lorsque les programmes deviendront un peu plus complexes.

Nous allons maintenant passer à l'étude de l'instruction IF... THEN. Elle est associée à un branchement, ou nœud de décision. Elle est symbolisée dans l'organigramme par un losange. Ce losange contient une question. L'embranchement suivi dépendra de la réponse à cette question : « oui » ou « non ».



Nœud de décision
L'instruction IF... THEN établit un branchement dans le programme.

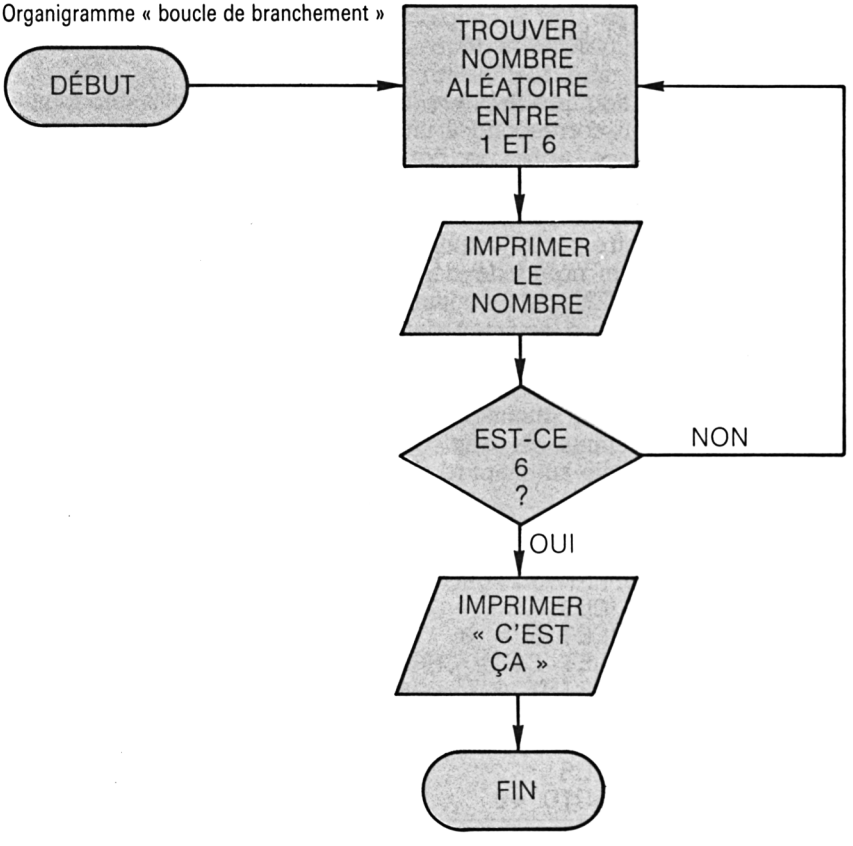
En voici l'application dans un programme. Nous l'utilisons ici pour ouvrir une boucle, mais l'on peut également s'en servir dans d'autres cas.

Essayez :

```
10 REM BRANCHEMENT
20 LET N=1 + INT(RND*6)
30 PRINT N
40 IF N=6 THEN GOTO 60
50 GOTO 20
60 PRINT "C'EST CA"
70 END
```

Voyez-vous ce qui se passe ici ? En effectuant des tours de boucle, le programme produit continuellement des valeurs différentes pour N (de la même manière que l'on jette plusieurs fois un dé). Si le nombre associé à N est différent de 6, l'ordinateur effectue un autre tour de boucle. Si la valeur obtenue est 6, alors l'instruction THEN agit et fait sortir l'ordinateur de la boucle.

Organigramme « boucle de branchement »



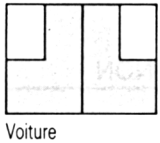
Remarquez que contrairement à la plupart des mots clés Basic, THEN n'est pas placée en début d'instruction ; THEN se situe au milieu de la ligne qui débute par IF et est normalement suivi d'une autre instruction qui indique à l'ordinateur ce qu'il doit faire si la condition IF est remplie. (Certains Basics omettent soit le THEN, soit

le deuxième mot de l'instruction, comme GOTO — mais, dans notre version, nous devons les faire figurer dans l'instruction.)

Examinons quelques exemples d'application. En voici un que vous rencontrerez souvent dans les listages de programmes de jeux :

```
100 REM BOUCLE POUR REJOUER
110 INPUT"VOULEZ-VOUS REJOUER (O/N)";R$
120 IF R$="O" THEN GOTO 30
130 END
```

Il s'agit de la phase finale d'une boucle plus importante qui correspond à une partie dans un jeu. Elle prend place après le corps du programme, et après le message préliminaire qu'il est inutile de répéter à chaque fois. C'est le contenu d'une variable de chaîne, entré par le joueur (R$, R pour réponse), qui donne lieu à un nœud de décision. Si (IF) R$ est « oui », alors (THEN) cela signifie que le joueur veut rejouer, et l'on entre à nouveau dans la boucle. Si (IF) R$ n'est pas « oui », alors (THEN) le programme quitte la boucle et parvient à son terme.



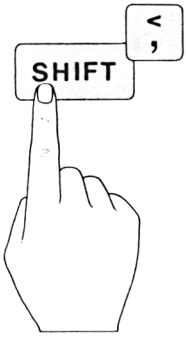
Voiture

Voici maintenant une façon de conclure le programme de déplacement horizontal que nous avons étudié au chapitre précédent.

Essayez :

```
10 REM *** VOITURE ***
15 CLS
20 LET P = 0
30 LET C$ = CHR$(136) + CHR$(132)
40 PRINT @ (10,P), C$
50 LET P = P + 1
60 IF P = 30 THEN GOTO 90
70 CLS
80 GOTO 40
90 GOTO 90
```

Cette fois, nous utilisons la variable P (position de la voiture) comme point de branchement. Si (IF) P = 30, ce qui veut dire que les deux caractères graphiques représentant la voiture occupent les deux dernières colonnes de notre écran (souvenez-vous, nous commençons à imprimer la voiture en P), alors (THEN) nous mettons un point final au programme. Si P est plus petit



que 30, alors, nous augmentons P et imprimons la voiture une colonne plus à droite.

Si nous utilisions un mouvement aléatoire pour la voiture, il faudrait alors apporter une légère modification. P pourrait en effet ne jamais être égal à 30. Un tour de boucle pourrait incrémenter P, par exemple, à 29, puis un suivant à 33. Afin de résoudre cette difficulté, nous demandons à l'ordinateur de vérifier si P est supérieur ou égal à 30. Il existe toute une série d' « opérateurs logiques » (en réalité des fonctions, tout comme les « opérateurs arithmétiques » +, −, etc.), utilisés par l'ordinateur pour déterminer si ce type de condition est remplie ou non. En voici l'énumération :

= égal à
< plus petit que
> plus grand que
< = plus petit ou égal à
> = plus grand ou égal à
< > différent de (plus grand ou plus petit).

Voici donc le programme voiture vu plus haut, mais adapté de manière à conférer à la voiture un mouvement aléatoire. Notez comment l'instruction IF a été transformée.

Essayez :

```
10 REM *** VOITURE ***
15 CLS
20 LET P = 0
30 LET C$ = CHR$(136) + CHR$(132)
40 PRINT @ (10,P), C$
50 LET P = P + R
60 IF P >= 30 THEN GOTO 90
70 CLS
80 GOTO 40
90 GOTO 90
```

Nous avons ajouté la ligne 45 afin de déplacer la voiture d'un nombre de colonnes aléatoire (de 1 à 3), à chaque fois que l'ordinateur effectue un tour de boucle. Nous avons modifié les lignes 50 et 60.

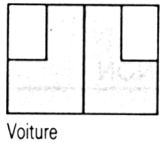
À la page suivante, vous trouverez l'organigramme de ce programme. Vous pourrez ainsi vous familiariser avec ce type de représentation.

le deuxième mot de l'instruction, comme GOTO — mais, dans notre version, nous devons les faire figurer dans l'instruction.)

Examinons quelques exemples d'application. En voici un que vous rencontrerez souvent dans les listages de programmes de jeux :

```
100 REM BOUCLE POUR REJOUER
110 INPUT"VOULEZ-VOUS REJOUER (O/N)";R$
120 IF R$="O" THEN GOTO 30
130 END
```

Il s'agit de la phase finale d'une boucle plus importante qui correspond à une partie dans un jeu. Elle prend place après le corps du programme, et après le message préliminaire qu'il est inutile de répéter à chaque fois. C'est le contenu d'une variable de chaîne, entré par le joueur (R$, R pour réponse), qui donne lieu à un nœud de décision. Si (IF) R$ est « oui », alors (THEN) cela signifie que le joueur veut rejouer, et l'on entre à nouveau dans la boucle. Si (IF) R$ n'est pas « oui », alors (THEN) le programme quitte la boucle et parvient à son terme.



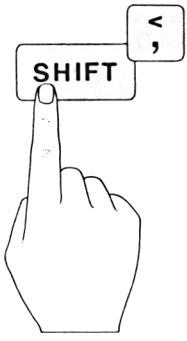
Voiture

Voici maintenant une façon de conclure le programme de déplacement horizontal que nous avons étudié au chapitre précédent.

Essayez :

```
10 REM *** VOITURE ***
15 CLS
20 LET P = 0
30 LET C$ = CHR$(136) + CHR$(132)
40 PRINT @ (10,P), C$
50 LET P = P + 1
60 IF P = 30 THEN GOTO 90
70 CLS
80 GOTO 40
90 GOTO 90
```

Cette fois, nous utilisons la variable P (position de la voiture) comme point de branchement. Si (IF) P = 30, ce qui veut dire que les deux caractères graphiques représentant la voiture occupent les deux dernières colonnes de notre écran (souvenez-vous, nous commençons à imprimer la voiture en P), alors (THEN) nous mettons un point final au programme. Si P est plus petit



que 30, alors, nous augmentons P et imprimons la voiture une colonne plus à droite.

Si nous utilisions un mouvement aléatoire pour la voiture, il faudrait alors apporter une légère modification. P pourrait en effet ne jamais être égal à 30. Un tour de boucle pourrait incrémenter P, par exemple, à 29, puis un suivant à 33. Afin de résoudre cette difficulté, nous demandons à l'ordinateur de vérifier si P est supérieur ou égal à 30. Il existe toute une série d' « opérateurs logiques » (en réalité des fonctions, tout comme les « opérateurs arithmétiques » +, −, etc.), utilisés par l'ordinateur pour déterminer si ce type de condition est remplie ou non. En voici l'énumération :

= égal à
< plus petit que
> plus grand que
< = plus petit ou égal à
> = plus grand ou égal à
< > différent de (plus grand ou plus petit).

Voici donc le programme voiture vu plus haut, mais adapté de manière à conférer à la voiture un mouvement aléatoire. Notez comment l'instruction IF a été transformée.

Essayez :

```
10 REM *** VOITURE ***
15 CLS
20 LET P = 0
30 LET C$ = CHR$(136) + CHR$(132)
40 PRINT @ (10,P), C$
50 LET P = P + R
60 IF P >= 30 THEN GOTO 90
70 CLS
80 GOTO 40
90 GOTO 90
```

Nous avons ajouté la ligne 45 afin de déplacer la voiture d'un nombre de colonnes aléatoire (de 1 à 3), à chaque fois que l'ordinateur effectue un tour de boucle. Nous avons modifié les lignes 50 et 60.

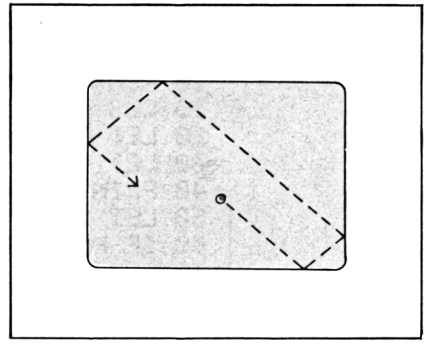
À la page suivante, vous trouverez l'organigramme de ce programme. Vous pourrez ainsi vous familiariser avec ce type de représentation.

La « super balle »

Jusqu'ici, dans les programmes que nous avons écrits, nous avons fait suivre THEN par GOTO. C'est ainsi, en effet, que l'on sort des boucles, que l'on arrête de tourner en rond et que l'on se raccorde à une autre partie du programme. Mais il est également possible d'utiliser IF... THEN à d'autres fins. Par exemple, nous pouvons faire suivre THEN d'une instruction LET qui changera la valeur d'une variable.

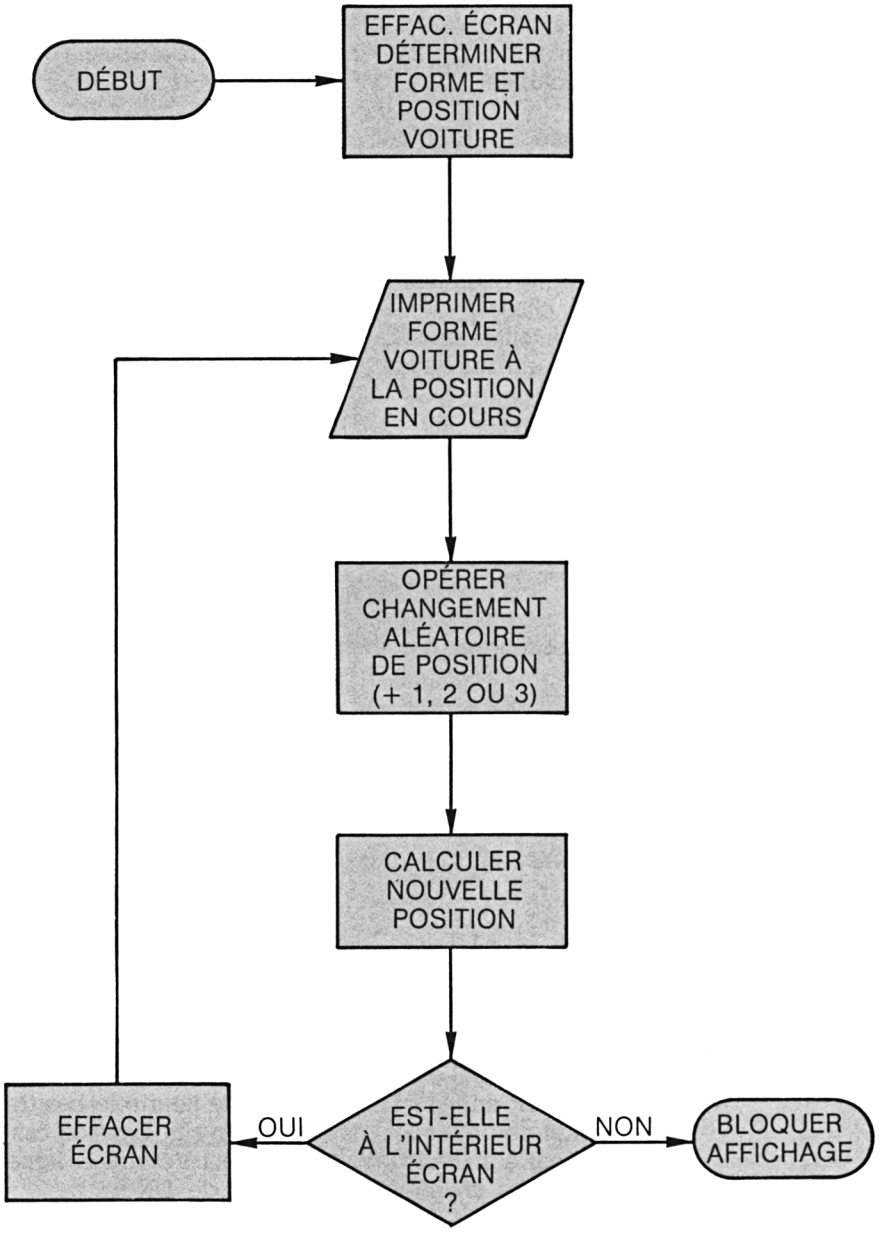
Passons à l'application pratique avec un nouveau programme : « la super balle ». Nous utiliserons un O en guise de balle. Mais peut-être préféreriez-vous un petit carré noir, un CHR$ (128)? L'idée du programme est très simple. Notre petite balle se déplace sur tout l'écran selon des trajectoires diagonales. À chaque fois qu'elle heurte le bord de l'écran, elle rebondit dans une autre direction. C'est exactement le même principe, enrichi de quelques fioritures, qui est utilisé dans des programmes de jeux qui simulent des parties de tennis. La programmation n'en est pas aussi simple qu'il y paraît. Aussi nous préparerons l'élaboration de notre programme à l'aide d'un organigramme (voir page suivante).

La « super balle »
Nous pouvons utiliser les nœuds de décision pour modifier la valeur d'une variable. Dans ce cas, le changement de coordonnées de la balle produira un changement de direction. La ligne pointillée montre de quelle façon la balle se déplace, rebondit sur le bord de l'écran pour repartir en sens inverse.



Nous aurons besoin de quatre variables, et non pas de deux comme vous le pensiez peut-être : une pour la rangée dans laquelle nous imprimerons la balle et une pour la colonne ; il nous en faudra une également pour la direction du mouvement horizontal (à gauche ou à

Organigramme « voiture »



droite) et une autre pour la direction du mouvement vertical (vers le haut ou vers le bas). Nous appellerons les deux premières R et C et les deux autres DR et DC, toutes étant des variables numériques, bien sûr.

Selon notre pratique habituelle, constituons la liste des variables :

Rangée	R
Colonne	C
Direction horizontale	DR
Direction verticale	DC

et écrivons quelques lignes de programme pour éclairer les choses.

Fixons tout d'abord la position initiale de la balle, par exemple au milieu de l'écran :

```
100 LET R = 8 : LET C = 15
```

Ensuite, changeons la position de la balle d'une rangée et d'une colonne à chaque tour de boucle :

```
110 LET DR = 1 : LET DC = 1
```

Pour descendre, par exemple, nous écrirons :

```
200 LET R = R + DR
```

de telle sorte que, si R était égal à 8, il devienne égal à 8 + 1 = 9.

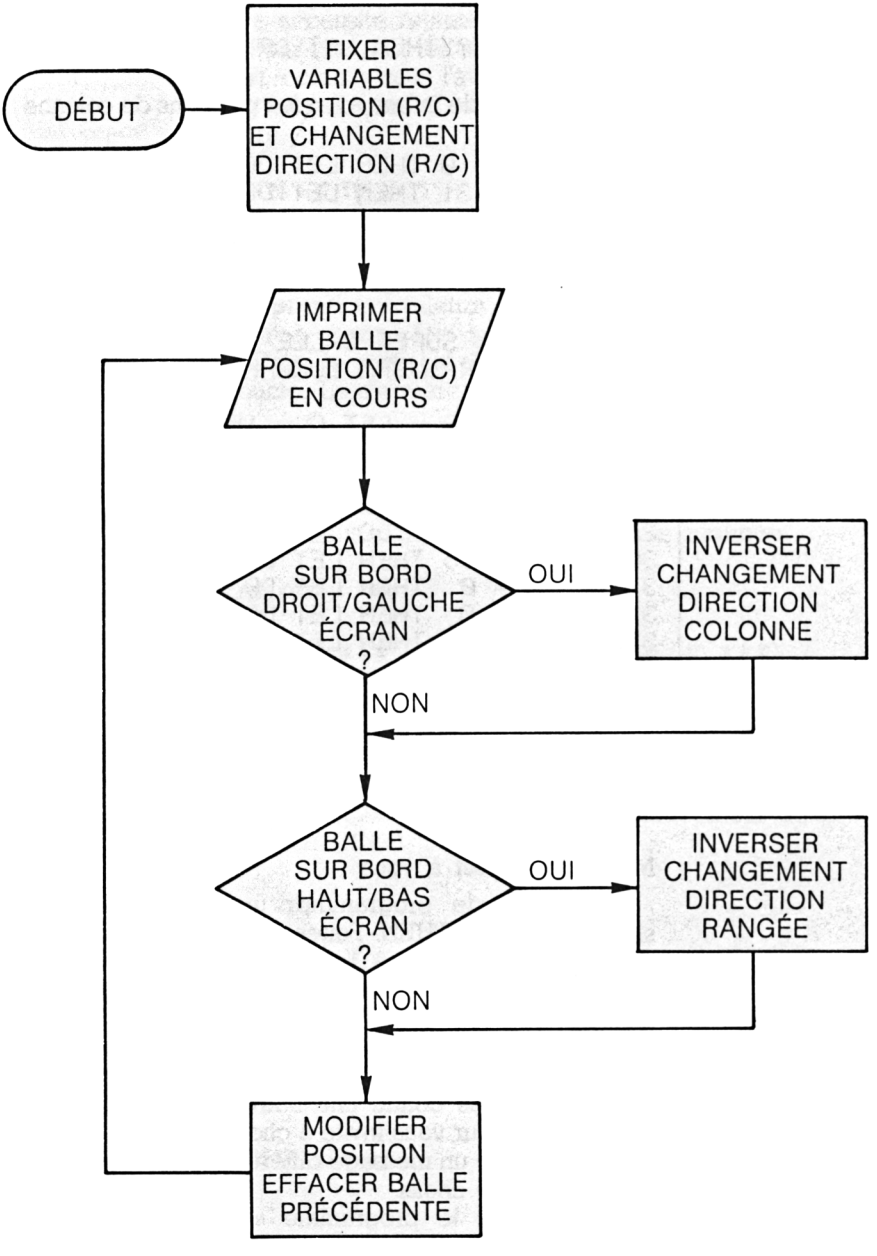
Pour monter, nous utiliserons simplement le nombre négatif − 1 :

```
LET DR = -1
```

(cela n'est pas à proprement parler une ligne, mais juste une instruction que nous devrons bientôt introduire dans le programme). Ainsi, si R était égal à 8, il deviendrait égal 8 + (−1) = 7. Mais quand faut-il changer DR en −1 ? Lorsque la balle heurte le bas de l'écran et ne peut aller plus bas, bien sûr. Dans ce cas, R est égal à 15. Ainsi, voici comment s'écrira la ligne lorsque nous changerons la valeur de DR :

```
300 IF R = 15 THEN LET DR = -1
```

Organigramme « super balle »



Lorsque la balle heurte le haut, naturellement, il faut procéder à la modification contraire :

```
310 IF R = 0 THEN LET DR = 1
```

Et nous faisons de même avec les positions de colonne :

```
320 IF C =  0 THEN LET DC =  1
330 IF C = 31 THEN LET DC = -1
```

Voilà l'essentiel de notre programme rédigé. Mettons-y la touche finale et voyons si vous vous y retrouvez :

```
10  REM *** SUPER BALLE ***
20  REM  PAR SUSAN CURRAN
30  CLS
100 LET R = 8 : LET C = 15
110 LET DR = 1 : LET DC = 1
120 PRINT @ (R,C), "O";
200 LET R = R + DR
210 LET C = C + DC
300 IF R = 15 THEN LET DR = -1
310 IF R = 0  THEN LET DR =  1
320 IF C = 0  THEN LET DC =  1
330 IF C = 31 THEN LET DC = -1
340 CLS
350 GOTO 120
```

La briéveté du programme a de quoi surprendre, non ? Reportez-vous à l'organigramme pour bien saisir la finalité de chaque ligne.

Maintenant, c'est à vous

1) Essayez de programmer un jeu de devinette simple avec IF... THEN. Faites choisir à l'ordinateur un nombre entre 1 et 6 à l'aide de la fonction RND et ensuite essayez de deviner ce nombre. Entrez votre réponse (utilisez INPUT). Si (IF) votre réponse est bonne, alors (THEN) faites en sorte que l'ordinateur vous le dise. Si vous vous êtes trompé, alors faites que l'ordinateur vous donne une autre chance.

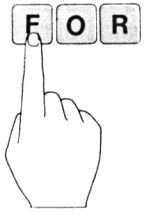
2) L'ordinateur vous invite à choisir un nombre entre 1 et 6 et affiche un message différent suivant le nombre que vous avez choisi.

3) Réécrivez le programme « Charmeur de serpent » de la page 65 en faisant s'immobiliser le serpent lorsqu'il atteint le haut de l'écran.

Compteur de boucle

Voici maintenant une autre technique utilisée dans les boucles ouvertes, celle du « compteur de boucle ». Elle nécessite l'utilisation de l'instruction FOR... NEXT.

Pour compter les nombres de tours de boucle, on se sert, comme vous l'avez peut-être deviné, d'une variable numérique. C'est le mot clé FOR qui intervient alors :




```
FOR N = 1 TO 10
```

Il est ainsi demandé à l'ordinateur de mettre en place un compteur de boucle — que nous appellerons N —, qui prend les valeurs 1, 2, 3... jusqu'à 10, successivement à chaque fois qu'un tour de boucle est effectué.

C'est le mot clé NEXT qui permet de fermer la boucle. La dernière ligne de la boucle s'écrira ainsi :


```
NEXT N
```

Encore une fois, ceci vous semblera sûrement plus clair quand nous passerons à l'application pratique.

Essayez :

```
10 REM *** COMPTEUR DE BOUCLE ***
20 FOR N = 1 TO 10
30 PRINT N
40 NEXT N
50 END
```

Voici ce que vous obtenez après lancement du programme (RUN) :

```
 1
 2
 3
 4
 5
 6
 7
 8
 9
 10
OK
```

Revenons un peu sur tout cela. La ligne 20 demande à l'ordinateur de produire une suite de nombres et de

se tenir prêt à les passer en revue un par un. Elle commence par donner à N la valeur du premier nombre de la suite. La ligne 40 renvoie à la ligne 20 qui attribue alors à N la valeur du second nombre de la suite, et ainsi de suite.

Notez que FOR et NEXT n'appartiennent pas à la même ligne de programme. Ils n'en constituent pas moins une seule et même instruction, comme IF... THEN. L'instruction FOR serait incomplète si elle n'était pas suivie d'un NEXT, situé quelque part dans le programme. Vous pouvez d'ailleurs insérer un grand nombre de choses bien plus compliquées dans la boucle FOR... NEXT, si vous le désirez. Vous trouverez ci-après un exemple un peu plus exotique...

Essayez :

```
10  REM *** PYRAMIDE ***
20  CLS
30  LET C = 15
40  LET A$ = "*"
50  FOR R = 0 TO 15
60  PRINT @ (R,C), A$;
70  LET C = C - 1
80  LET A$ = A$ + "**"
90  NEXT R
100 GOTO 100
```

Aucune récompense particulière à la clé cette fois-ci si vous devinez le résultat obtenu sur l'écran!

Vous pouvez voir combien FOR... NEXT s'avère utile pour le comptage des boucles, et c'est elle que l'on utilise en général lorsque l'on veut qu'un programme effectue une opération un certain nombre de fois avant de passer à autre chose. Néanmoins, parfois, il n'est pas nécessaire de prolonger le comptage jusqu'à son terme. Par exemple, dans un jeu, on peut laisser six tentatives à un joueur pour répondre à une question, ou bien pour atteindre une cible. S'il réussit en moins de six essais, alors il faut sortir de la boucle pour pouvoir afficher : « gagné »; dans ce cas, FOR... NEXT n'est pas réellement appropriée.

La plupart des ordinateurs ne vous préviennent pas immédiatement que vous avez commis une erreur lorsque vous sortez prématurément d'une boucle FOR... NEXT; mais recourir à ce subterfuge relève d'un manque de respect pour les règles élémentaires de la

bonne programmation. L'ordinateur garde une trace des FORs et de leurs correspondants NEXTs dans sa mémoire, et s'il n'en épuise pas le stock lorsqu'il exécute le programme, alors cette dernière sera surchargée d'entrées indésirables. Si l'on veut sortir prématurément d'une boucle, il faut utiliser la construction IF... THEN, exactement comme nous l'avons fait dans le programme « VOITURE » à la page 85. C'est un peu plus long, mais cela évite que des problèmes ne surgissent à long terme.

Voici un exemple de ce type de boucle. C'est un jeu de devinette ; le programme est semblable à celui que nous vous avions demandé d'effectuer à la page 90. Vous verrez que nous utilisons un compteur, T, pour le nombre de tentatives dont le joueur dispose, mais qu'au lieu de le combiner avec FOR, nous l'associons à LET.

Essayez :

```
10  REM *** DEVINETTE ***
20  LET N = 1 + INT(RND*10)
30  PRINT "DEVINEZ LE NOMBRE"
40  PRINT "AUQUEL JE PENSE."
50  PRINT "SI VOUS NE TROUVEZ PAS ";
55  PRINT "EN 5 ESSAIS,"
60  PRINT "JE VOUS DONNERAI LA REPONSE."
70  LET T = 1
80  INPUT G
90  IF G = N THEN GOTO 180
100 LET T = T + 1
110 IF T = 5 THEN GOTO 140
120 PRINT "NON ! ESSAYEZ ENCORE."
130 GOTO 80
140 PRINT "DESOLE, C'ETAIT VOTRE"
150 PRINT "DERNIERE TENTATIVE."
160 PRINT "LA REPONSE ETAIT ";N;"."
170 END
180 PRINT "BRAVO ! C'ETAIT ";N;"."
190 END
```

Ce programme a une structure assez complexe, aussi nous vous en donnons l'organigramme. Remarquez les deux fins possibles. Si le joueur ne trouve pas la réponse en cinq essais, alors l'ordinateur exécutera les lignes 140 à 160 et n'ira pas plus loin. Si le joueur trouve la bonne réponse, alors les lignes 170 et 180 fourniront la conclusion du programme.

Boucles emboîtées

Les boucles deviennent un outil encore plus efficace si nous les « emboîtons » — c'est-à-dire si nous incluons de petites boucles dans de plus grandes. Néanmoins, il faut se garder de les emmêler. Il convient de s'assurer qu'après avoir ouvert une boucle se trouvant à l'intérieur d'une autre, on l'a bien refermée, et ce, toujours à l'intérieur de la seconde boucle.

Bien sûr, si nous utilisons des boucles de comptage, nous devons établir deux compteurs et leur donner des noms de variable différents. Voici un exemple.

Essayez :

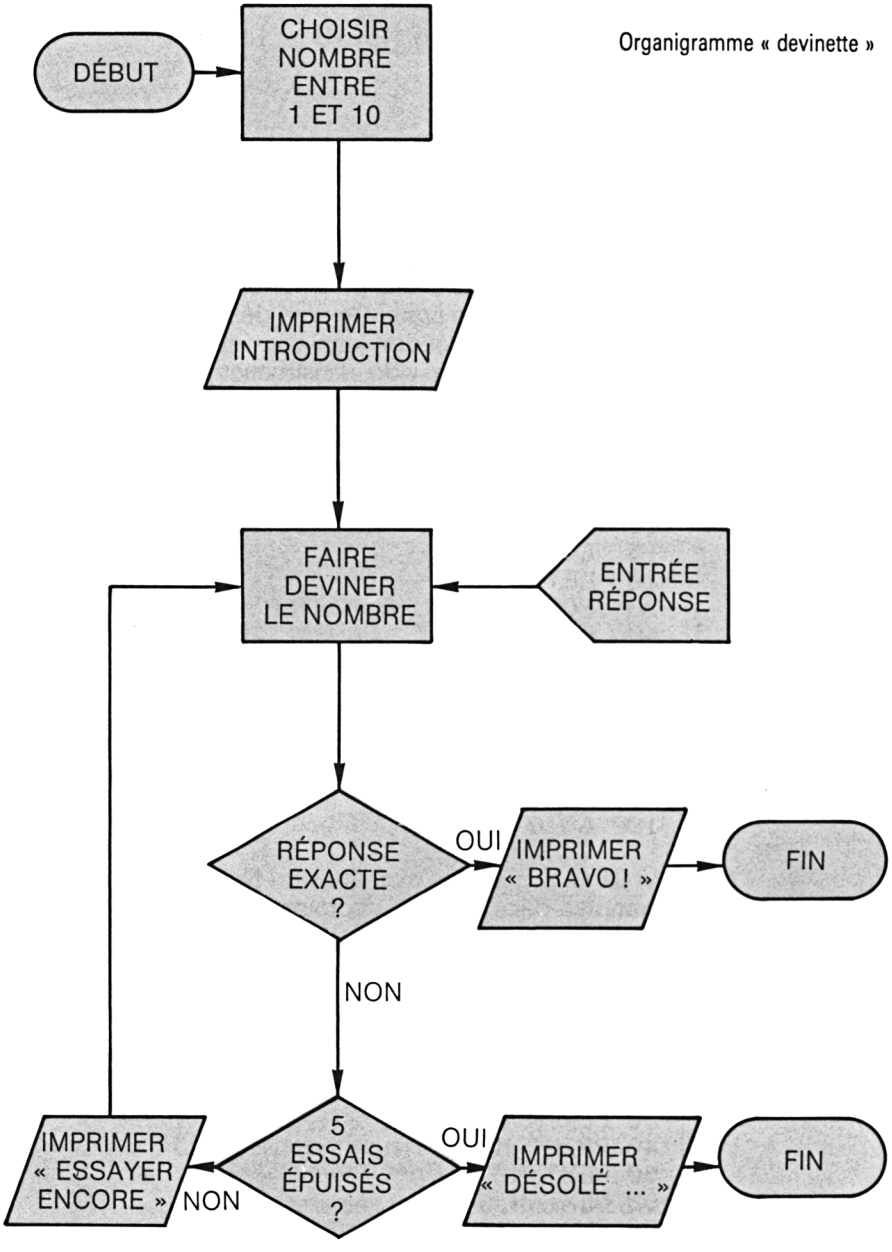
```
10 REM *** HORLOGE ***
20 PRINT "JE COMPTERAI JUSQU' A 10"
30 PRINT "PAR INTERVALLES D'UNE SECONDE"
40 FOR N = 1 TO 10
50 PRINT N
60 FOR D = 0 TO 460
70 NEXT D
80 NEXT N
90 END
```

Notez la façon dont nous avons repéré les boucles, à la gauche du programme, afin de nous assurer qu'elles sont bien emboîtées.

Vous serez peut-être intrigués par la boucle D intérieure. Que contient-elle ? Rien ! L'ordinateur passe 460 fois de la ligne 60 à la ligne 70. Vous souvenez-vous de notre robot clignotant du chapitre 2 ? Nous tirions parti du temps nécessaire à l'ordinateur pour exécuter nos commandes et l'utilisions comme un chronomètre qui réglait la fréquence de clignotement du nom du robot. Cette fois-ci, nous nous servons également de la boucle vide comme d'une minuterie. Il faut environ 1 seconde à notre ordinateur pour effectuer les 460 tours de notre boucle — d'où le titre du programme. Dorénavant, nous choisirons la lettre D comme nom de compteur dans le cas d'une « boucle de temporisation ». De cette façon vous saurez reconnaître une telle boucle, chaque fois que nous en utiliserons une.

Introduisons un raffinement supplémentaire qui nous permettra de passer à un autre programme. Il s'agit du mot clé STEP, associé à l'instruction FOR... NEXT. Dans les exemples que nous avons donnés jusqu'ici, FOR était utilisé pour faire compter l'ordinateur, unité par unité : 1, 2, 3, etc. En fait, nous pouvons lui dire de compter en progressant à chaque fois d'une quantité

quelconque (nombres entiers, décimaux, positifs ou négatifs), placée après STEP.



Organigramme « devinette »

Essayez :

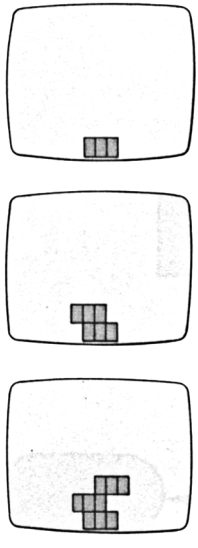
```
10 REM *** HORLOGE ***
20 FOR N = 0 TO 60 STEP 5
30 PRINT N
40 FOR D = 0 TO 460*5 : NEXT D
50 NEXT N
60 END
```

Cette fois l'horloge fonctionne à la même vitesse, mais elle n'imprime un nombre que toutes les 5 secondes. Et nous devons modifier la boucle de temporisation en conséquence. Nous n'avons pas besoin de calculer 460 fois 5, puisque l'ordinateur peut le faire pour nous!

Cette fois, nous avons placé la boucle de temporisation sur une seule ligne. Ce sera désormais notre façon habituelle de procéder (afin d'insister sur le fait qu'il s'agit d'une boucle vide d'instruction).

Voici deux nouveaux exemples dans lesquels le compteur est utilisé à des fins différentes :

Reptations
Ce « serpent » se déplace en rampant vers le haut de l'écran d'une rangée à la fois, de façon aléatoire.



Essayez :

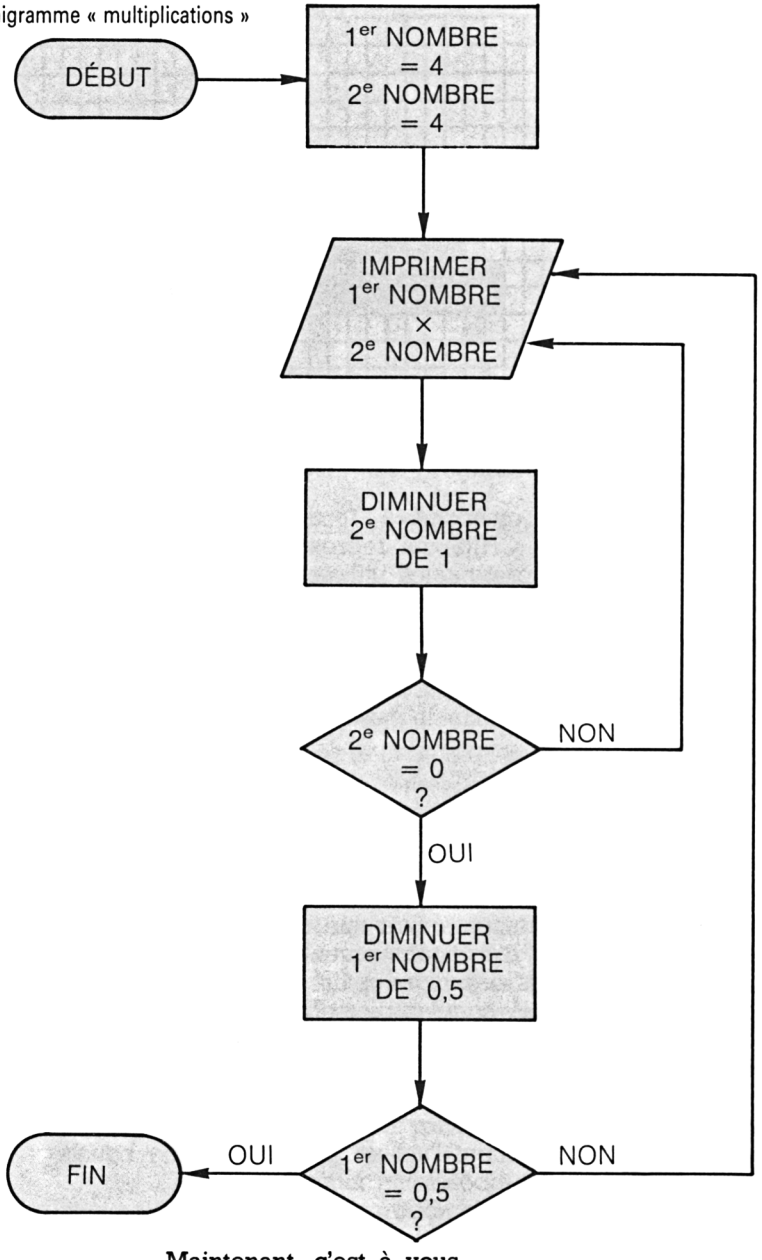
```
10  REM *** REPTATIONS ***
15  CLS
20  LET C = 15
30  FOR R = 15 TO 0 STEP -1
40  PRINT@ (R,C), CHR$(128)+CHR$(128)+
    CHR$(128);
50  LET C = C - 2 + INT(RND*5)
60  IF C < 0 THEN LET C = 0
70  IF C > 29 THEN LET C = 29
80  FOR D = 0 TO 500: NEXT D
90  NEXT R
100 GOTO 100
```

Voilà un reptile des plus évolués! Notez que si l'on veut compter dans le sens décroissant, il faut faire suivre STEP d'un nombre négatif.

Essayez :

```
10  REM *** MULTIPLICATIONS ***
15 CLS
20 FOR N = 4 TO 1 STEP -0.5
30 FOR M = 4 TO 1 STEP -1
40 PRINT N;"X";M;"=";N*M;
50 NEXT M
60 NEXT N
70 END
```

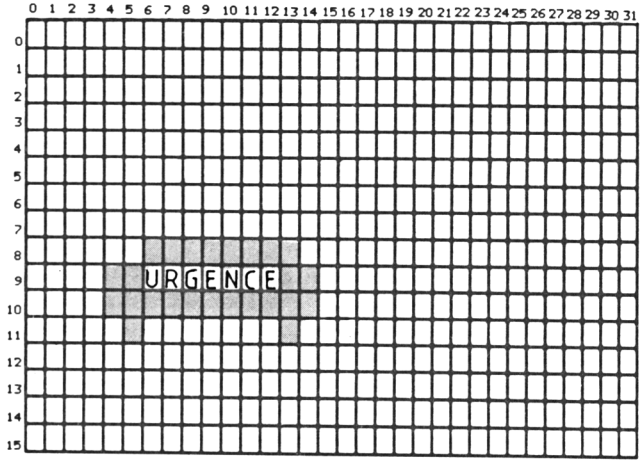
Organigramme « multiplications »



Maintenant, c'est à vous

4) Dessinez une ambulance sur l'écran, avec le mot « URGENCE » clignotant lentement sur le côté.

Ambulance
Voici notre suggestion pour le dessin d'une ambulance munie d'une enseigne clignotante.



5) Demandez à l'ordinateur d'imprimer tous les codes CHR$ qui représentent l'ensemble des caractères graphiques utilisés par notre ordinateur.

6) Programmez une minuterie. Faites en sorte que l'ordinateur vous demande combien de secondes il lui faudra compter, puis qu'ensuite il les compte une par une, jusqu'à ce qu'il atteigne le nombre indiqué. Il affichera alors le message suivant : « TEMPS ÉCOULÉ ».

Contrôle

Arrivé à la fin de ce chapitre, vous devriez savoir :

Sur les boucles
— quelle est la différence entre une boucle ouverte et une boucle fermée ;
— comment ouvrir une boucle à l'aide d'un compteur, ou d'un branchement.

Sur les organigrammes
— quels sont les symboles de base d'un organigramme et comment les utiliser.

Sur l'instruction IF... THEN
— comment l'utiliser pour constituer un nœud de décision.
— comment l'utiliser pour ouvrir une boucle.

Sur les opérateurs logiques
— ce qu'ils sont, et leur utilisation élémentaire.

Sur l'instruction FOR... NEXT
— comment elle est utilisée, notamment pour programmer une boucle de retard ;
— comment utiliser le mot clé STEP avec FOR... NEXT.

5

Édition et débogage

Au cours de ce chapitre, nous étudierons :
— les erreurs de programmation (bogues) et leur élimination,
— les possibilités d'édition sur les ordinateurs domestiques,
— les procédures de contrôle

Nous allons marquer une pause dans l'écriture des programmes afin d'aborder un sujet très important : comment faire en sorte que les programmes fonctionnent !

Si vous avez déjà étudié les chapitres 3 et 4, travaillé nos exemples et réalisé les programmes présentés à la rubrique « maintenant, c'est à vous », vous avez certainement rencontré des difficultés lorsqu'il s'est agi de faire fonctionner ces programmes (il en va de même pour tous les programmeurs !). Notre but, dans ce chapitre, est de vous apprendre à résoudre ces problèmes et de vous démontrer combien il est important de savoir « éditer » et « déboguer » correctement.

Éditer un programme, c'est tout simplement procéder à des corrections, apporter des améliorations à ce qui a déjà été tapé. Bogue (en anglais, bug) est le mot usuel désignant tout ce qui est de nature à contrarier le fonctionnement de votre programme. Il est très rare que votre ordinateur lui-même soit à l'origine d'un mauvais fonctionnement. La plupart du temps, c'est vous qui avez commis une erreur ; il apparaît en fait que vous n'avez pas écrit le programme que vous pensiez écrire. Quelle que soit la cause de l'erreur, il faut « déboguer » pour que le programme puisse « tourner » correctement.

Si vous ne possédez pas encore d'ordinateur, vous vous sentirez moins concerné par ces considérations. Lisez cependant ce chapitre ; les sujets traités sont importants et ils contribueront à mieux vous faire saisir ce qu'est la programmation.

Regarder les choses en face

Dans beaucoup d'activités, il est préférable d'oublier les erreurs le plus rapidement possible. On efface tout et l'on recommence. Il n'en va pas de même en programmation. Il est vain d'espérer devenir un bon programmeur si l'on se contente de taper NEW à chaque fois qu'un programme ne marche pas. Il faut tirer la leçon de ses erreurs en épluchant les listages des programmes défectueux, en tentant d'y apporter des solutions et en appliquant une méthode qui permette systématiquement de dépister les bogues et d'y remédier.

Avez-vous commis beaucoup d'erreurs jusqu'à présent ? Ou êtes-vous susceptible d'en commettre à l'avenir ? Tout cela dépend de votre méthode de travail. Certains préfèrent peaufiner leur programme avant de le taper. Ensuite, ils le vérifient ligne par ligne, avant de l'exécuter. Il arrive tout de même qu'ils n'obtiennent pas ce qu'ils désirent, mais c'est rare. D'autres privilégient l'approche rapide et brouillonne. Ils exécutent le programme aussitôt qu'ils ont écrit quelques lignes et attendent que l'ordinateur leur indique l'endroit où ils ont commis des erreurs.

Quelle est la meilleure approche ? Les deux ont leurs avantages et leurs inconvénients. La plupart des ordinateurs, même les plus petits et les plus rudimentaires, sont capables de détecter les erreurs et d'aider les utilisateurs à éditer les lignes de programmes sans difficulté. Autant exploiter ces capacités. Une ligne de Basic incorrectement formulée n'a jamais endommagé un ordinateur ! Au pire, il sera pris dans une boucle sans fin, qu'il sera possible d'interrompre soit en appuyant sur la touche BREAK, soit en coupant l'alimentation de l'ordinateur ! Au nombre des avantages, figure aussi la possibilité de détecter facilement une erreur lors de l'exécution du programme : une partie de l'affichage sur l'écran peut vous paraître erronée, par exemple, ou bien un message d'erreur apparaît. Sur le papier, un point-virgule oublié se remarque difficilement. C'est perdre son temps que de consacrer une heure au contrôle de chaque signe de ponctuation avant d'exécuter le programme. L'ordinateur est là pour

vous aider — pour peu que vous lui en laissiez la possibilité.

D'un autre côté, il est également important de vous assurer que vous avez parfaitement construit votre programme. S'il est bien conçu, il restera efficace même entre les mains les moins expertes.

Il apparaît donc que l'un des aspects importants de l'élaboration soigneuse d'un programme consiste en la mise en place d'une procédure de vérification et de débogage. Il faut en fait partir du présupposé que le programme ne marchera pas, et effectuer toutes les vérifications en conséquence. C'est un conseil qui n'est pas aussi négatif qu'il y paraît ! Il se révélera des plus efficaces à chaque fois qu'il s'agira de modifier ou d'améliorer un programme existant.

Voici quelques-unes des lignes directrices qui doivent vous guider :

1) Subdivisez le programme en petites parties. Généralement, il vous sera facile de déterminer les articulations où vous pourrez procéder aux coupes. Celles-ci interviendront lors du passage à l'exécution d'une tâche nettement différente de celle qui la précède : début d'un calcul complexe, dessin d'un nouveau graphique, contrôle des réponses d'un joueur, etc. Vous rappelez-vous la façon dont nous avons subdivisé le programme « maison » au chapitre 2 ? Introduisez chaque nouveau sous-ensemble par une REMarque explicative et, pourquoi pas, par un numéro de ligne distinctif.

2) Gardez beaucoup de numéros de ligne disponibles, afin d'être en mesure de pouvoir insérer de nouvelles lignes entre celles qui existent déjà, si le besoin s'en fait sentir. C'est particulièrement important si votre ordinateur ne possède pas de commande RENUM — commande extrêmement précieuse puisqu'elle permet d'aérer un programme en renumérotant les lignes (par exemple, dans un programme surchargé, les lignes 10, 11, 12, 13... peuvent être renumérotées 10, 20, 30, 40..., etc.

3) Contrôlez le programme autant que possible, partie après partie. Ainsi vous pourrez localiser d'éventuelles erreurs avant d'exécuter le programme. Par exemple, vérifiez sur l'écran les formes que vous avez conçues, avant de les faire s'animer d'une façon complexe. Contrôlez le corps principal du programme, calculs ou autres, avant de mettre en place les parties introductrices ou finales. Vous pouvez mettre en mémoire quelques portions du programme en même

temps, puis les contrôler une par une, en utilisant des techniques simples pour les isoler. Pour contrôler les lignes 50 à 100 d'un programme par exemple, on pourra faire intervenir les instructions suivantes :

```
2   GOTO 50
101 END
```

qu'il sera ensuite facile d'effacer en tapant seulement leurs numéros de ligne suivis d'un blanc :

```
2   {E}
101 {E}
```

Nous vous avons déjà fourni quelques lignes directrices quant à la manière de planifier la réalisation d'un programme préalablement à sa codification en Basic : établissement d'une liste des variables, élaboration d'un organigramme, etc. Bien évidemment, il vous faudra les expérimenter pour découvrir la méthode qui vous convient. Mais, que faire lorsque, malgré tout, l'on continue à rencontrer des difficultés ?

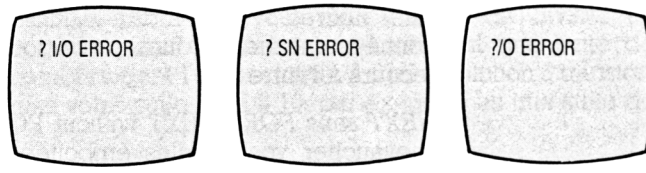
Cela dépend en partie de l'aide que votre ordinateur est en mesure de vous apporter. Envisageons trois possibilités.

1. Vous essayez d'exécuter un programme et un message d'erreur apparaît.

Voici au moins un problème qui se pose en termes clairs. L'ordinateur vous informe que quelque chose ne va pas et, avec un peu de chance, il vous dira précisément de quoi il s'agit. Tout ce que vous avez à faire est de jeter un coup d'œil là où l'ordinateur vous le demande et d'apporter la correction requise.

Les messages d'erreur varient beaucoup selon l'ordinateur. Le nôtre produit de brefs messages, tels "SN Error" ou "TM Error" que vous connaîtrez bientôt. D'autres donnent tout simplement un numéro de code : "Error 42", par exemple. Il faut consulter un tableau dans le manuel, qui fournit la signification de "Error 42" (erreur de syntaxe, par exemple). Certains ordinateurs disposent de toute une gamme de messages ; d'autres, au contraire, n'en possèdent qu'un nombre réduit et de portée générale. La plupart des ordinateurs vous indiqueront quelle ligne de votre programme les a arrêtés, certains même diront quelle est l'instruction qu'ils refusent dans cette ligne.

Messages d'erreur
La façon d'indiquer la nature de l'erreur varie avec les ordinateurs. La figure vous montre les messages d'erreurs courants qui apparaissent sur notre écran.



Erreur d'entrée/sortie | Erreur de syntaxe | Division par zéro

Si l'erreur se trouve au milieu du programme, alors il vaut mieux — en attendant d'être plus expérimenté — la corriger avant d'exploiter le programme. Sinon, l'ordinateur s'arrêtera net à chaque fois qu'il atteindra l'instruction qu'il ne comprend pas. Si l'erreur est située tout à la fin, vous pouvez, si vous le voulez, l'ignorer, comme nous l'avons fait dans certains de nos premiers exemples. La conclusion du programme sera un peu bancale, mais celui-ci fonctionnera parfaitement jusqu'à ce qu'il arrive à son terme.

Voici quelques messages d'erreur courants que vous rencontrerez sûrement, si ce n'est pas déjà fait !

Appel de fonction erroné ("Illegal function call" en anglais) ; ce message peut apparaître si vous avez utilisé, par exemple, un nombre qui n'appartient pas à l'éventail utilisé pour la désignation des positions d'impression : l'ordinateur ne peut se rendre à la ligne 17 sur un écran de seize lignes ! La solution : assurez-vous que votre programme contienne une procédure permettant de vérifier que les nombres associés à une variable sont acceptables et d'agir si tel n'est pas le cas. (Voyez à ce sujet le programme « super balle » à la page 88.)

Erreur d'entrée/sortie ("Input/Output error") ; ce message vous indique que le programme ne s'est pas chargé correctement ou que l'un des périphériques d'entrée/sortie (par exemple le clavier) ne fonctionne pas comme il le devrait. Beaucoup d'interfaces à cassette (l'ensemble des circuits qui permettent à l'ordinateur d'utiliser un lecteur de cassette) sont capricieuses ; il vous faudra peut-être plusieurs tentatives, en choisissant à chaque fois un réglage du volume légèrement différent, avant que le programme ne se charge correctement. Vérifiez, bien sûr, que vous avez choisi le bon côté de la cassette ! Et faites deux copies de tous les programmes importants, de sorte que si une copie

est « altérée » (c'est-à-dire inutilisable parce que les données ont été perdues) vous pourrez toujours avoir recours à l'autre.

NEXT sans FOR ("NEXT without FOR"); vous avez fait chevaucher vos boucles emboîtées; vérifiez que les NEXT et les FOR se correspondent, en usant de notations simples comme celles que nous employons à la gauche de nos programmes au chapitre 4.

Erreur de syntaxe ("Syntax error"); mise en garde très générale qui vous avertit d'une erreur commise dans la façon dont vous avez écrit vos instructions Basic; par exemple :
— vous avez peut-être mal orthographié un mot clé, et écrit, par exemple, PRTIN au lieu de PRINT;
— vous avez peut-être oublié un signe de ponctuation essentiel, comme les deux points qui séparent deux instructions sur la même ligne.

Erreur d'affectation ("Type mismatch"); vous avez essayé d'associer une chaîne de données à une variable numérique ou vice versa, et ceci soit directement, soit dans le cadre d'une instruction qui utilise les contenus de variables. Voici un exemple d'erreur :

```
LET C$=T+1
```

(C$ est un nom de variable de chaîne, et T est un nom de variable numérique). Vous devez remédier à cette erreur en changeant votre nom de variable ou votre instruction, afin que variables de chaîne et variables numériques soient utilisées de façon cohérente. Parfois, il ne s'agit que d'une erreur de frappe. Plus grave, il arrive que vous ayez tenté une manipulation que l'ordinateur n'a pas accepté; dans ce cas, vous découvrirez peut-être que l'une des fonctions de manipulation de chaînes dont votre version Basic dispose (pouvant servir à transformer les variables numériques en variables de chaînes, etc.) permet de vous aider à résoudre le problème auquel vous vous trouvez confronté. A nouveau, c'est votre manuel qui vous fournira tous les renseignements concernant ces fonctions.

Ligne inconnue ("Undefined line"); ce message apparaît lorsque vous essayez d'aller (GOTO) à un numéro de ligne que vous n'avez pas utilisé dans votre programme. La solution : modifiez l'instruction GOTO en conséquence.

Division par zéro; vous avez essayé de diviser un nombre par 0 (sans doute en utilisant une suite de nombres dont l'un était égal à 0). La solution : transformez votre suite de 0 à 10, par exemple en une suite de 1 à 11.

2. Vous essayez d'exécuter le programme et votre sortie n'est pas satisfaisante.

L'ordinateur ne se soucie pas de vérifier si ce que vous lui demandez est logique ; il s'assure seulement qu'il est capable de réaliser ce que vous exigez de lui. Souvent, donc, votre programme ne suscitera aucune contestation de la part de l'ordinateur, mais vous n'obtiendrez pas le résultat escompté. Voici le genre de problèmes qui peuvent surgir :

— Alors que vous dessiniez un objet mobile, vous n'avez pas effacé les positions précédentes. Aussi vous obtenez une série de fragments d'automobile, par exemple, d'un bout à l'autre de l'écran, au lieu d'une seule voiture, entière.
— Vous avez interverti vos numéros de ligne et de colonne ou vous avez tapé des codes CHR$ erronés et, en conséquence, l'image obtenue est totalement fantaisiste.
— Vous avez omis une parenthèse essentielle dans une ligne contenant des formules mathématiques complexes. Aussi l'ordinateur n'a pas effectué les calculs auxquels vous vous attendiez.
— Vous avez mal tapé un nom de variable, et l'ordinateur a compris que vous faisiez référence à une autre variable et lui a donné la valeur 0.

Dans tous ces cas il vous appartient de déceler l'instruction posant problème. Si vous avez documenté votre programme en y insérant beaucoup de REMarques, la tâche n'en sera que plus aisée. Nous étudierons cela plus en détail au chapitre 7.

3. Vous exécutez le programme et rien ne se passe.

Voilà bien la situation la plus délicate. En effet, l'ordinateur ne peut dans ce cas vous être d'un grand secours. Appuyez sur la touche BREAK et voyez où le programme s'arrête. Vous obtiendrez sûrement un message du type :

```
BREAK IN LINE 60
```

Cela peut vous fournir une indication. Si la touche BREAK ne fonctionne pas, consultez votre manuel pour remettre le système à zéro. De nombreux ordinateurs vous offrent cette possibilité, sans effacer pour autant votre programme. Si rien n'y fait, vous pouvez toujours débrancher la prise de courant, mais vous perdrez alors toutes les informations contenues dans la machine.

Comment peut-il ne rien se passer? Il est possible que vous ayez simplement oublié de demander à l'ordinateur d'afficher le résultat. (Dans ce cas, vous avez reçu le message OK vous indiquant que l'ordinateur a obéi à toutes vos instructions.) Mais le plus souvent, l'ordinateur se sera perdu dans une boucle sans fin, effectuant indéfiniment la même instruction sans produire de sortie.

Comment dépister l'erreur? L'une des solutions consiste à entrer des lignes d'indication de parcours. Supposons que vous vous attendiez à ce que l'ordinateur imprime un petit bonhomme aux lignes 500 à 600; insérez alors des lignes de programme supplémentaires, par exemple :

```
501 PRINT "LIGNE 501"
521 PRINT "LIGNE 521"
```

De cette façon, vous saurez si l'ordinateur se rend aux lignes 500, 520 ou autres. Peut-être avez-vous oublié un GOTO, peut-être certaines parties de votre programme ont-elles mal été construites ou avez-vous mal emboîté vos boucles.

Vous verrez que les techniques que nous vous avons présentées, telle la division d'un programme en sous-parties, se révèlent bien commodes dans ce cas. L'utilisation de GOTO et de END ou des commandes équivalentes dans votre version Basic s'avère précieuse pour le contrôle de sous-parties isolées, ou pour le court-circuitage de portions de programme que l'on soupçonne d'être fautives. Il suffit ensuite de vérifier si le reste du programme fonctionne.

Techniques d'édition

Les possibilités d'édition varient encore plus d'un ordinateur à l'autre que les messages d'erreurs. Certains d'entre eux par exemple, après avoir détecté une erreur à l'intérieur d'une ligne, réécrivent automatiquement cette ligne à l'endroit où se trouve le curseur, et la tiennent prête à être éditée et corrigée. D'autres disposent de programmes d'édition (éditeurs) pleine

page grâce auxquels il est possible de passer rapidement le listage en revue, de bas en haut, à l'aide du curseur, jusqu'à ce que l'on parvienne à l'endroit voulu afin d'y apporter des corrections. D'autres disposent seulement d'éditeurs de ligne. Il faut alors utiliser une commande du type :

```
EDIT 50
```

avant de modifier la ligne 50. Si votre ordinateur possède une commande EDIT, consultez votre manuel afin d'apprendre à l'utiliser pour les corrections.

Sur certains ordinateurs, il est facile de modifier le texte d'une ligne déjà entrée en utilisant les commandes spéciales d'édition et en déplaçant le curseur. Sur d'autres, vous n'avez pas d'autre alternative que de retaper entièrement la ligne.

Si les capacités d'édition de ligne de votre ordinateur sont médiocres, essayez d'écrire de courtes lignes à instruction unique, comportant peu de caractères. De cette façon, la corvée de réécriture des lignes sera réduite au strict minimum. Si vous voulez vous débarrasser d'une seule ligne, tapez simplement le numéro qui la désigne et appuyez sur ENTER. Certains ordinateurs ont une commande DELETE qui permet de supprimer toute une partie du programme en une seule fois.

Si vous avez beaucoup d'éditions à faire, alors n'oubliez pas que votre programme, même s'il ne fonctionne pas, peut toujours être conservé sur bande ! Ainsi, vous pouvez interrompre votre travail d'édition pour y revenir plus tard. Si vous effectuez un travail important sur un programme long, il vous faudra préserver le listage au fur et à mesure pour parer à tout incident qui provoquerait la perte de la version se trouvant dans l'ordinateur.

Sachez vous avouer vaincu

Grâce aux techniques d'édition, vous pouvez transformer complètement un programme déjà entré dans l'ordinateur. Toutefois, même un gros travail d'édition ne pourra vous tirer d'affaire si vous avez commis une erreur fondamentale dans la conception de votre programme.

Prenez le temps de réfléchir quand le message d'erreur apparaît ; il est vain de tenter toute une suite de modifications dans l'espoir d'en découvrir une qui, par chance, marchera. Parfois, vous trouverez la solution exacte qui permettra de faire tourner le pro-

gramme. Et d'autres fois, vous accepterez le fait qu'il n'existe pas de solution, et que vous vous êtes tellement égaré que le plus sage consisterait à tirer un trait sur tout ce qui a déjà été fait, pour repartir à zéro.

Peu d'ordinateurs savent tout faire. Il vous arrivera de découvrir que votre machine ne peut réaliser la tâche qui lui aura été soumise. Probablement parce que cette tâche dépasse ses capacités de traitement : sa mémoire, par exemple, s'avère insuffisante ou il ne dispose pas des caractères dont vous avez besoin. Ces insuffisances peuvent également avoir pour cause les limites imposées par la version Basic utilisée ; il faut alors recourir à un autre langage de programmation (si votre machine le permet). Nous aborderons ce sujet au chapitre 8.

Testez vos programmes

Lorsque vous avez eu des difficultés pour faire tourner un programme, c'est vraiment un grand soulagement quand il finit par faire ce que vous lui avez demandé. Néanmoins, ce n'est pas encore fini. Vous devez vous assurer que ce programme fonctionne dans tous les cas, et non seulement pour l'exemple que vous avez choisi ! Un programme peut-il fonctionner une fois, et pas deux ? Bien sûr ; il vous faut notamment examiner les points suivants :

1. Le programme fonctionne-t-il lorsque l'on entre des données différentes ?

Si l'utilisateur peut choisir entre des nombres allant de 0 à 10, le programme fonctionnera-t-il s'il tape 0 ? 10 ? Fonctionnera-t-il pour n'importe quel nombre ? Si vous donnez à l'utilisateur la possibilité d'entrer son nom, le programme tournera-t-il, quel que soit la longueur du nom ? Essayez différentes possibilités, ou testez toutes les possibilités si vous avez établi une liste de choix. Dans le cas où vous voulez déplacer des objets sur l'écran, se produira-t-il quelque chose lorsqu'ils heurteront le bord de l'écran ; votre programme ne sera-t-il pas pris en défaut ? Si vous n'avez pas prévenu l'utilisateur qu'il ne faut choisir que des nombres entiers, le programme fonctionnera-t-il s'il prend des nombres décimaux ?

2. Que se passe-t-il lorsque vous n'entrez pas les bonnes données ?

Jusqu'à présent, nous n'avons pas incorporé beaucoup de procédures de contrôle dans nos programmes. Si

nous avons prévu de faire entrer par l'utilisateur des nombres de 0 à 10, nous avons de bonnes raisons de penser qu'il choisira effectivement des nombres compris entre 0 et 10. Cependant, il est toujours souhaitable de prévoir une espèce de sous-programme qui vérifie que l'utilisateur entre effectivement une donnée compatible et le re-sollicite dans le cas contraire.

Il existe un grand nombre de techniques permettant d'incorporer des sous-programmes de contrôle, mais beaucoup d'entre elles sont trop compliquées pour être étudiées ici. En voici deux illustrations simples :

```
40 PRINT "CHOISISSEZ UN NOMBRE ENTRE
   5 ET 50"
50 INPUT N
60 IF N < 5 THEN GOTO 40
70 IF N >50 THEN GOTO 40
```

Celle-ci est évidente. Il est demandé à l'utilisateur d'entrer un second nombre si le premier fourni ne se situe pas dans la fourchette désirée.

```
100 INPUT "VOULEZ-VOUS JOUER A NOUVEAU
    O/N";R$
110 IF R$ = "O"   THEN GOTO 30
120 IF R$ = "OUI" THEN GOTO 30
```

Nous sollicitons la réponse « O », mais si l'utilisateur répond « OUI », il pourra rejouer quand même. Des techniques plus sophistiquées pourraient accepter, par exemple, n'importe quel mot commençant par « O » ou « o » comme s'il s'agissait de « OUI ».

D'autres techniques de contrôle permettent de garantir, par exemple , que l'utilisateur fournit la quantité exacte de lettres ou de nombres demandée. Par exemple, pour une date, on pourra vérifier que celle-ci est plausible.
Pour un nombre, on pourra contrôler qu'il est entier et non pas décimal, etc.

3. Toutes les boucles fonctionnent-elles correctement ?

Prenons à nouveau l'exemple d'un programme de jeu. Ce n'est pas parce que le sous-programme qui produit le « VOUS AVEZ PERDU » fonctionne, qu'il en est de même de celui qui produit le « VOUS AVEZ GAGNÉ » ! Quant au sous-programme commandant le « REJOUEZ », renvoie-t-il de façon logique à un endroit

précis du jeu? Toutes les variables ont-elles été re-initialisées, autorisant ainsi une nouvelle partie? Au cours de cette nouvelle partie, le score est-il affiché correctement?

Il est certain que ces procédures de contrôle ne sont pas aussi passionnantes que l'écriture d'un programme. C'est une corvée, mais une corvée nécessaire. Si vous voulez écrire des programmes que votre famille, vos amis ou d'autres personnes puissent également utiliser, alors il faut vous accoutumer à ces contrôles. Une fois l'habitude prise, vous y trouverez un autre avantage : celui de savoir utiliser et modifier les listages de programmes qui vous sont présentés dans les livres ou dans les magazines. Certains d'entre eux, et l'on peut le déplorer, contiennent des erreurs dues soit à la négligence de leurs auteurs, soit à des erreurs typographiques. Lors de longues saisies, vous ne pourrez vous-même éviter des erreurs de frappe! Il vous faudra donc bien apprendre à détecter et à corriger ces erreurs.

Dans le chapitre 7, nous étudierons tous les aspects de l'écriture de deux programmes longs. Nous reviendrons à cette occasion sur certaines des questions abordées ici. Nous examinerons le type de fautes que vous êtes susceptibles de commettre, et envisagerons quelques-unes des possibilités de correction.

Contrôle

Arrivé à la fin de ce chapitre, vous devriez savoir :

Sur le débogage

— comment se présentent les messages d'erreur sur votre ordinateur, et comment en trouver la signification dans votre manuel;
— comment utiliser des techniques simples pour le dépistage et la suppression des erreurs dans vos programmes.

Sur l'édition

— de quelles ressources votre ordinateur dispose et comment les utiliser;
— si les commandes EDIT et RENUM sont disponibles sur votre ordinateur;
— pourquoi il est important de contrôler un programme en totalité, et comment réaliser des contrôles simples.

6

Manipulation de données

Au cours de ce chapitre, nous étudierons :
— les tableaux de données à une dimension,
— l'instruction DIM,
— l'instruction READ... DATA.

Nous avons déjà eu un aperçu du rôle des variables dans la manipulation des données. Il nous faut maintenant aborder différentes méthodes qui permettent d'utiliser un bien plus grand nombre de données que nous ne l'avons fait jusqu'ici. Grâce à ces méthodes, nous introduirons dans le programme, au moment de son élaboration ou de son exécution, une liste de données à exploiter.

Si l'on veut n'utiliser qu'une donnée à la fois, alors il est possible (comme vous le savez désormais) de n'avoir qu'une seule variable, et de lui associer successivement des contenus différents. Pour vous rafraîchir la mémoire, jetez un coup d'œil à ce court programme :

Essayez :

```
10  REM *** FAHRENHEIT ***
20  PRINT "DONNEZ-MOI"
30  PRINT "UNE TEMPERATURE FAHRENHEIT"
40  PRINT "ET JE VOUS DONNERAI SON"
50  PRINT "EQUIVALENT EN CENTIGRADES"
60  PRINT "ENTREZ 0 POUR ARRETER"
70  INPUT "FAHRENHEIT";F
80  IF F = 0 THEN GOTO 120
90  PRINT "L'EQUIVALENT EN CENTIGRADES"
100 PRINT "DE ";F;"EST ";(F-32)*5/9
110 GOTO 70
120 END
```

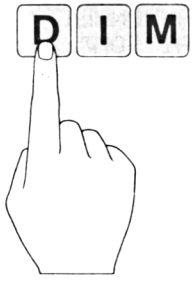
Nous n'utilisons ainsi qu'une variable F pour toutes les températures Farenheit que l'utilisateur voudra entrer. Remarquez de quelle façon nous amenons le programme à sa fin : nous avons recours à une valeur, F = 0, que l'utilisateur a peu de chance de vouloir normalement entrer. Cela nous permet d'ouvrir la boucle et d'arriver sur END.

Cette méthode convient parfaitement tant que le programme ne permet d'entrer qu'une ou deux données, et qu'il les exploite avant d'autoriser à en entrer d'autres. Mais elle s'avérera inefficace si l'on désire entrer beaucoup de données afin que le programme les utilise toutes en une seule fois, ce qui est souvent ce que l'on recherche, par exemple, pour trier une liste.

Bien sûr, il est toujours possible d'introduire dans le programme une longue liste de noms de variables A, B, C, D, etc., et de faire en sorte que l'ordinateur les utilise une par une. Toutefois, il ne sera pas facile de programmer ainsi une boucle. Alors comment faire ?

Prenons un exemple tiré du quotidien : les jours de la semaine. Nous pouvons en établir une liste que nous appellerons « JOUR » et ensuite affecter un numéro à chacun d'entre eux : 1 pour lundi, 2 pour mardi, etc. Nous pouvons utiliser la variable J$ pour représenter la liste et la variable N pour la valeur des numéros. Allons plus loin et examinons le programme suivant :

```
10  REM ***INTRODUCTION AUX TABLEAUX***
20  DIM A$(5)
30  PRINT "ENTREZ 5 VILLES"
40  PRINT "ET JE LES LISTERAI"
50  FOR N = 1 TO 5
60  INPUT "VILLE :";A$(N)
70  NEXT N
80  REM MAINTENANT LA LISTE
90  FOR N = 1 TO 5
100 PRINT N,A$(N)
110 NEXT N
120 END
```



Une nouvelle instruction a été introduite dans ce programme, à la ligne 20. L'aviez-vous remarquée ? DIM signifie DIMension. Nous informons l'ordinateur de la dimension d'un ensemble de données ; en d'autres termes, nous lui indiquons le nombre d'emplacements à prévoir. Un tel ensemble s'appelle un « tableau ». Vous remarquerez que les emplacements ont des noms qui ressemblent beaucoup à ceux que nous avons déjà

utilisés pour les variables. On place le signe $ après le nom d'une variable de chaîne exactement comme nous l'avons fait jusqu'à présent. Mais, cette fois, l'on fait suivre le signe $ par un nombre entre parenthèses. Les différentes villes du tableau auront ainsi des noms individualisés :

```
A$(1)
A$(2)
A$(3)
```

et ainsi de suite, jusqu'à la valeur maximale que nous avons fixée : A$(5) dans notre exemple.

L'instruction DIM indique donc à l'ordinateur quelle est cette valeur maximale, de telle sorte qu'il puisse aménager l'espace nécessaire, dans sa mémoire pour stocker les informations du tableau. Quelquefois, avec certains Basics, l'ordinateur effectue cette opération automatiquement. Notre ordinateur s'attend, lui, à voir apparaître une commande DIM à chaque fois que nous constituons un tableau.

Notre ordinateur numérote le tableau à partir de 1 (1, 2, 3, et ainsi de suite). Certains ordinateurs débutent à 0, mais il n'est pas nécessaire d'apporter des modifications aux programmes que nous vous proposons si vous vous trouvez dans ce cas : il vous suffira de ne pas affecter de valeur à la variable 0.

On se sert de tableaux chaque fois que l'on dispose d'un ensemble de données de la même espèce. Ainsi on peut facilement concevoir, comme nous l'avons déjà vu, un tableau contenant les jours de la semaine et qui ressemblerait à ceci :

```
LET J$(1) = "LUNDI"
LET J$(2) = "MARDI"
```

Cependant, le réel avantage de tels tableaux est de permettre le recours à une variable pour désigner le numéro des articles contenus dans le tableau. Dans notre exemple, nous avons utilisé N. C'est une variable totalement autonome, qui prendra successivement les valeurs 1, 2, 3, etc., à chaque tour d'une boucle et à chaque entrée d'une valeur différente dans le tableau.

Bien évidemment, il est recommandé de procéder comme nous l'avons fait, c'est-à-dire d'utiliser la même variable à la fois pour compter notre boucle à l'aide de l'instruction FOR... NEXT et pour désigner les numéros des articles de notre tableau. De même, après avoir mis en place une première boucle pour entrer toutes

les données, nous en avons utilisé une seconde pour imprimer la liste de ces données. Afin de ne pas compliquer notre programme, nous avons seulement imprimé le tableau ; mais nous aurions pu faire des choses beaucoup plus complexes, à partir du moment où tout était soigneusement stocké dans la mémoire de l'ordinateur.

Passons à un autre exemple. Cette fois, nous mettrons en place une boucle permettant d'entrer une série de positions d'impression de carrés graphiques ; ou, pour être plus précis, deux séries de positions : l'une pour les rangées et l'autre pour les colonnes. Nous constituerons deux tableaux, R et C, dans lesquels viendront prendre place les données.

Ils devront indiquer à l'ordinateur combien de données nous placerons dans chaque tableau ; mais cette fois, le nombre de données pourra varier en fonction du nombre de carrés désiré. Accordons-nous un maximum de 50 couples de positions d'impression. Si nous préférons en utiliser moins, nous pourrons toujours laisser vide une partie du tableau.

Nous obtenons ainsi une première ligne :

```
DIM R(50) : DIM C(50)
```

(Pas de $ cette fois puisque ce sont des nombres qui vont constituer le tableau.)

Comme nous ne savons pas à l'avance de combien de couples nous nous servirons, nous n'aurons pas recours à FOR... NEXT, mais utiliserons une boucle formée avec GOTO et ouverte par IF... THEN. Deux situations permettront d'ouvrir la boucle : d'une part l'entrée d'une valeur particulière (1 000) indiquant que l'on a fini d'introduire les données et, d'autre part, le dépassement de la dimension autorisée. Il est facile de pourvoir à ces deux cas de figure.

Comme nous n'aurons pas recours cette fois à un compteur de boucle, nous utiliserons une variable N pour déterminer à quel numéro nous sommes parvenus dans chacun des tableaux. Aussi, nous devrons incrémenter cette variable d'une unité à chaque tour de boucle.

Voici le programme : Vous en trouverez l'organigramme à la page 116.

Essayez :

```
10  REM ***DESSIN***
20  DIM R(50):DIM C(50):LET N=1
30  PRINT "INDIQUEZ-MOI OU PLACER"
40  PRINT "LES CARRES SUR L'ECRAN"
45  PRINT "ET JE DESSINERAI POUR VOUS"
50  PRINT "50 CARRES MAXIMUM"
60  PRINT "TAPEZ 1000 POUR FINIR"
70  REM . COLLECTE DES DONNEES
80  INPUT "POSITION DE RANGEE";R(N)
90  IF  R(N) = 1000 THEN GOTO 200
100 INPUT "POSITION DE COLONNE";C(N)
110 LET N = N + 1
120 IF N = 51 THEN GOTO 200
130 GOTO 80
200 REM DESSIN
210 CLS
220 FOR M = 1 TO (N-1)
230 PRINT@ (R(M),C(M)),CHR$(128);
240 NEXT M
250 GOTO 250
```

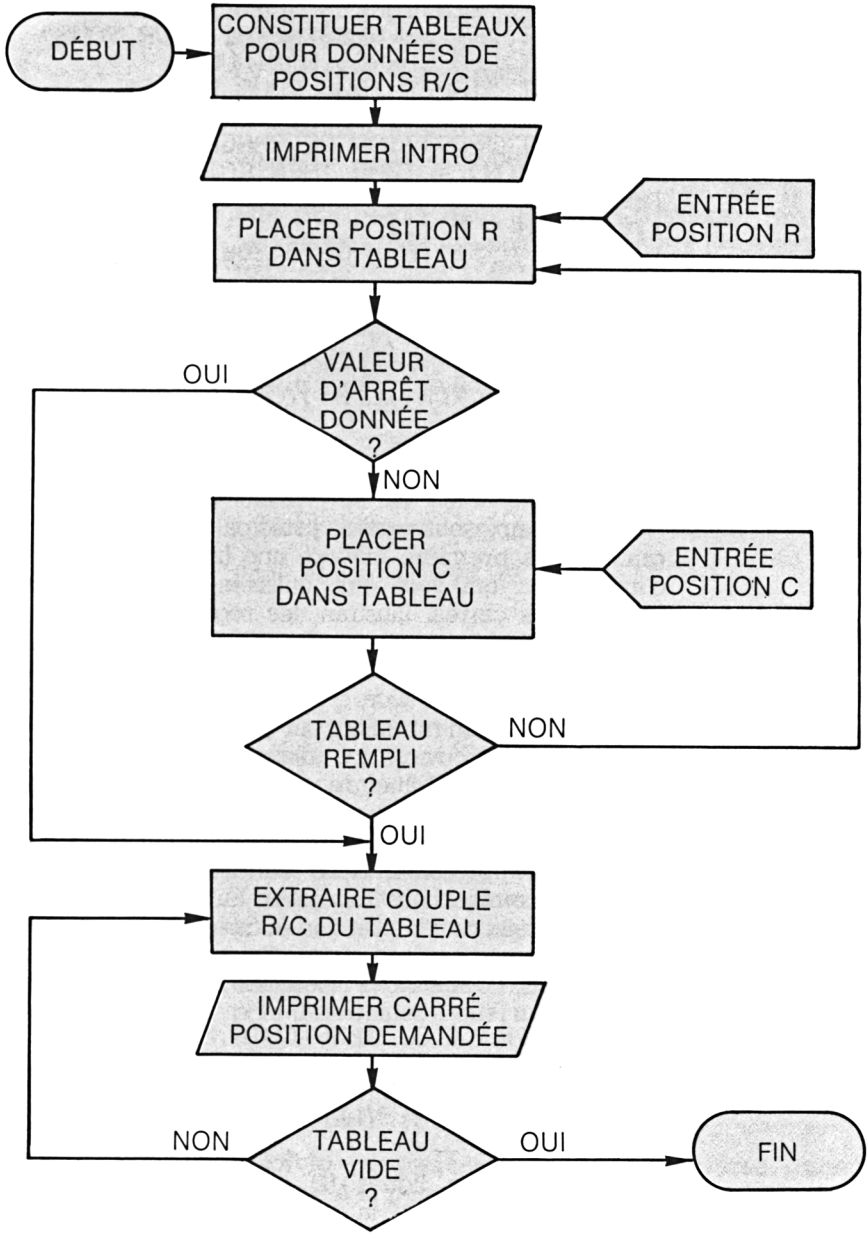
Il y a deux sous-parties principales dans le programme. La première élabore une liste des positions d'impression qui vont être utilisées, et la seconde imprime les carrés. Illustrant les règles exposées au chapitre précédent, nous avons clairement séparé ces sous-parties à l'aide de REMarques et de numéros de ligne distinctifs.

Notez que, contrairement au programme "FARENHEIT" nous ne pouvions ici utiliser le 0 comme valeur d'arrêt, car il est habituel de vouloir imprimer un carré à la position (0, 0). Aussi, nous avons utilisé 1 000 qui se situe largement en dehors de l'éventail utilisable des positions d'impression. Nous aurions pu également contrôler, comme nous l'avons vu au chapitre précédent, les positions d'impression entrées par l'utilisateur, afin de s'assurer à chaque fois qu'elles sont recevables. Dans ce cas, nous aurions inséré les lignes suivantes :

```
92  IF R(N) < 0  THEN GOTO 80
94  IF R(N) > 15 THEN GOTO 80
102 IF C(N) < 0  THEN GOTO 100
104 IF C(N) > 31 THEN GOTO 100
```

En revenant aux lignes 80 et 100 (INPUT), nous réécrirons simplement la nouvelle information « par-dessus » l'information erronée.

Organigramme « dessin »



Comme la notion de tableau est neuve pour vous, et comme elle vous déroute peut-être, lançons ce programme pour voir ce qui se passe. Vous pourrez avec profit examiner à la fois l'organigramme et les lignes du programme. Cela vous aidera à comprendre comment chacune d'elles fonctionne.

D'abord, bien sûr, l'ordinateur imprimera le message d'introduction, nous informant de ce qu'il veut de nous. Ensuite, il sollicitera l'entrée des données. Demandons-lui d'imprimer trois carrés au milieu de l'écran, aux positions (8, 10), (8, 11) et (8, 12). Voici l'échange qui en résultera :

l'ordinateur	*l'utilisateur*
POSITION DE RANGEE ?	8
POSITION DE COLONNE ?	10
POSITION DE RANGEE ?	8
POSITION DE COLONNE ?	11
POSITION DE RANGEE ?	8
POSITION DE COLONNE ?	12
POSITION DE RANGEE ?	1000

Au quatrième tour de boucle, nous entrons la valeur d'arrêt 1 000 et l'ordinateur passe alors à la seconde partie du programme.

Voici les fichiers de données qui figurent alors dans la mémoire de l'ordinateur :

	emplacement	*contenu*
tableau R :	R(1)	8
	R(2)	8
	R(3)	8
	R(4)	1 000
	R(5 à 50)	0
tableau C :	C(1)	10
	C(2)	11
	C(3)	12
	C(4 à 50)	0
variable N :		4

Remarquez que N est maintenant supérieur d'une unité aux nombres de carrés que nous voulons imprimer, parce que nous utilisons également cette variable pour la valeur d'arrêt. Aussi, dans la deuxième partie du programme, à ligne 220, nous demandons à l'ordinateur de prendre dans les tableaux les données dont les numéros varient de 1 à N-1.

Nous employons une deuxième variable, M, qui joue le rôle de compteur pour la seconde partie du programme. L'ordinateur effectue trois tours de boucle. Il va chercher à chaque tour les coordonnées de position de rangée et de colonne d'un carré dans les deux tableaux R et C. C'est compris ? Nous l'espérons, car...

Maintenant, c'est à vous

1) Écrivez un programme qui vous demande d'entrer quatre nombres et les imprime ensuite sur une colonne, au milieu de l'écran.

2) Écrivez un programme qui vous demande d'entrer jusqu'à 20 codes CHR$ et les imprime ensuite sur une rangée, au milieu de l'écran.

3) Écrivez un programme qui vous demande d'entrer cinq noms et les imprime ensuite en ordre inverse.

Si vous vous sentez encore mal à l'aise avec les tableaux, regardez les réponses page 189 et étudiez attentivement les programmes suggérés, et ce, ligne par ligne. Ces exercices devraient vous familiariser avec les tableaux.

Lecture des données

Nous sommes désormais capables d'utiliser les instructions d'entrée et les boucles répétitives pour introduire des données dans des tableaux. Mais il existe une autre méthode qui permet d'intégrer à un moment précis, des données dans un programme. Elle fait penser à un tableau qui fonctionnerait avec l'instruction LET.

Vous pouvez bien sûr utiliser l'instruction LET pour mettre en place des données dans un tableau, mais puisque vous ne pouvez placer les données que dans un seul emplacement à la fois, cela nécessiterait un grand nombre de LET, par exemple :

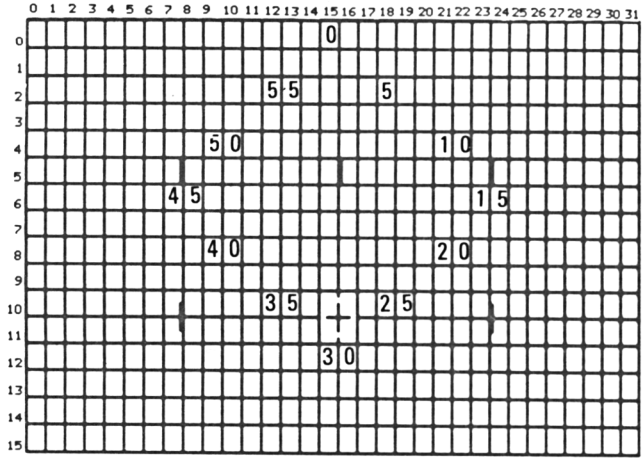
```
LET A(1) = 20
LET A(2) = 30
LET A(3) = 40
...
```

Aussi la méthode que nous abordons nous amène à introduire une nouvelle instruction : READ... DATA. Nous remplaçons en quelque sorte LET par READ, pour indiquer à l'ordinateur quels sont les emplacements auxquels nous allons associer des données, et le signe = par DATA, pour attribuer à chaque emplacement sa donnée. L'exemple suivant nous montre comment cette instruction fonctionne. Notre ordinateur va

jouer les horloges en imprimant 0 puis les multiples de 5 jusqu'à 60, selon une disposition à peu près circulaire qui évoquera un cadran d'horloge. Il lui faudra une minute pour afficher le cadran en entier. Les données (DATA en anglais) que nous introduirons dans le programme sont des numéros de rangée et de colonne, définissant treize positions (dont l'une est utilisée deux fois) et que l'ordinateur lira (READ en anglais) numéro par numéro. Observez l'organigramme et le croquis de l'horloge pour mieux comprendre ce que nous faisons.

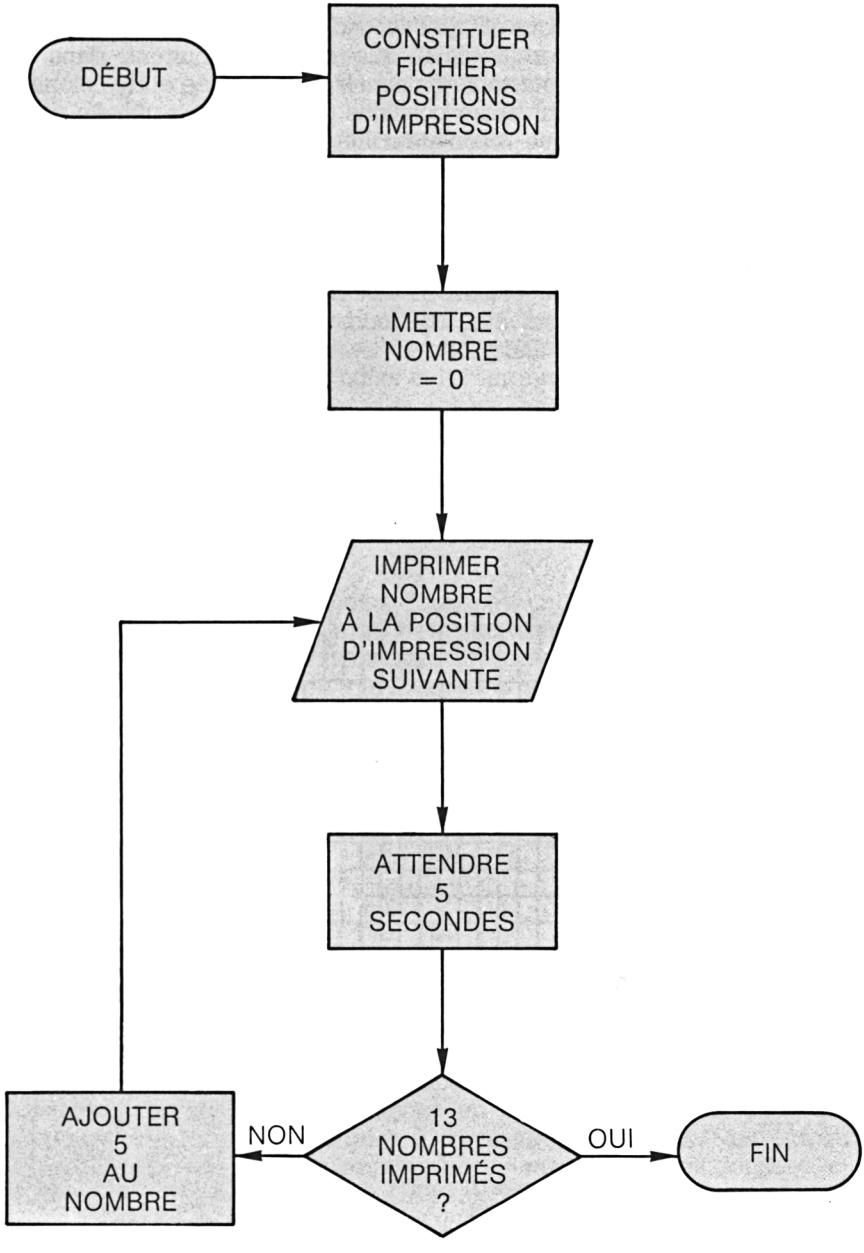
Il faudra fournir à l'ordinateur une position de rangée, puis une position de colonne, puis une autre position de rangée, etc., tout comme dans le précédent programme "DESSIN" :

rangée 0, colonne 15,
rangée 2, colonne 18,
rangée 4, colonne 21,
rangée 6, colonne 23,



L'horloge
Voici le croquis illustrant la sortie obtenue par le programme de la page 121. Les nombres apparaîtront, chacun leur tour, à cinq secondes d'intervalle.

Organigramme
« horloge »



Voici la mise en forme du programme.

Essayez :

```
10  REM ***HORLOGE***
20  REM TABLEAUX
30  DIM R(13):DIM C(13)
40  FOR N = 1 TO 13
50  READ R(N):READ C(N)
60  NEXT N
70  REM IMPRESSION TOUTES LES 5 SEC
80  CLS
90  FOR N = 1 TO 13
100 PRINT@ (R(N),C(N)),(N-1)*5;
120 FOR D = 0 TO 460*5: NEXT D
130 NEXT N
140 GOTO 140
200 DATA 0,15,2,18,4,21,6,23,8,21
210 DATA 10,18,12,15,10,12,8,9,6,7
220 DATA 4,9,2,12,0,15
```

Cette fois, il y a en fait trois sous-parties dans le programme. La première distribue les données dans les deux tableaux, le premier regroupant les positions de rangée et le second celles de colonne. Ces positions sont celles où viendront s'afficher les chiffres du cadran de l'horloge. La seconde partie imprime les chiffres de l'horloge toute les cinq secondes (rappelez-vous de la boucle de temporisation du chapitre 4). La troisième, enfin, fournit les données.

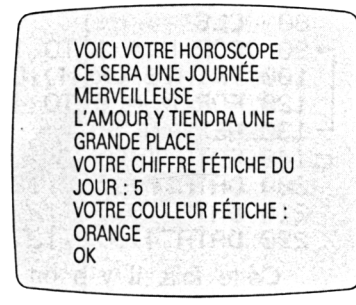
Vous devriez être à même de comprendre le processus par lequel l'instruction READ passe en revue les emplacements des tableaux, en demandant à l'ordinateur d'associer à chacun d'entre eux une donnée. Mais comment l'instruction DATA fonctionne-t-elle ? La meilleure façon de comprendre est de se représenter les données qui suivent DATA comme une longue liste d'articles que l'ordinateur passe en revue. Chaque fois que l'ordinateur rencontre READ, il lit une donnée dans la liste, dans l'ordre où elle se présente ; et ainsi de suite jusqu'à épuisement. Aussi n'y a-t-il pas besoin d'information supplémentaire pour indiquer à l'ordinateur quelle donnée est à associer à tel emplacement du tableau ; l'ordre des données dans la liste y supplée.

S'il y a trop de données dans votre liste, l'ordinateur ignorera celles qui sont en surnombre. S'il en manque, il émettra un message d'erreur : "OD" (Out of Data en anglais, c'est-à-dire : à court de données).

Remarquez que les données ont été placées après la boucle fermée qui clôt le programme ; l'ordinateur y

aura tout de même accès. En fait, il est possible de placer les données à peu près n'importe où dans le listage, que ce soit avant ou après les instructions READ. Et l'on peut utiliser l'instruction DATA pour associer des données à autant de variables, dans autant de tableaux qu'on le désire.

L'ordinateur vous donne votre horoscope
Ci-dessous : exemple typique d'un programme d' « horoscope ».



Voici un autre exemple :

```
 10  REM ***HOROSCOPE***
 15  REM TABLEAUX
 20  DIM A$(3):DIM B$(4):DIM C$(6)
 30  FOR X = 1 TO 3: READ A$(X):NEXT X
 40  DATA "MERVEILLEUSE","ENNUYEUSE",
     "DETESTABLE"
 50  FOR X = 1 TO 4 :READ B$(X):NEXT X
 60  DATA "L'ARGENT","LE TRAVAIL",
     "LA FAMILLE","L'AMOUR"
 70  FOR X = 1 TO 6: READ C$(X):NEXT X
 80  DATA "ROUGE","JAUNE","VERT","BLEU",
     "ORANGE","VIOLET"
 100 REM AFFICHAGE HOROSCOPE
 110 CLS
 120 PRINT@ (1,4),"VOICI VOTRE HOROSCOPE"
 130 PRINT@ (3,1),"CE SERA UNE JOURNEE ";
     A$(1+INT(RND*3))
 140 PRINT@ (5,1),B$(1+INT(RND*4));
     " Y TIENDRA UNE GRANDE PLACE"
 150 PRINT@ (7,1),"VOTRE CHIFFRE FETICHE
     DU JOUR: ";1+INT(RND*9)
 160 PRINT@ (9,1),"VOTRE COULEUR FETICHE
     : ";C$(1+INT(RND*6))
 170 END
```

Apparemment, il y a trois listes de données dans le programme, une pour chaque tableau : A$, B$ et C$. En fait, il n'y en a qu'une seule ; la ligne 130 effectue un choix entre les trois premiers articles de la liste, la ligne 140 entre les quatre suivants et la ligne 160 entre les six derniers. Pour mieux s'en rendre compte, changeons la ligne 40 en :

```
40 DATA "MERVEILLEUSE","ENNUYEUSE",
   "DETESTABLE","PASSIONNANTE"
```

En procédant à quelques exécutions ou en demandant à l'ordinateur d'imprimer les contenus des tableaux, on se rendra compte alors que la dernière donnée de la première instruction DATA est placé dans le deuxième tableau, etc.

Comme on le voit, il est nécessaire de bien contrôler les instructions DATA, spécialement lorsqu'elles contiennent des listes de nombres, afin de s'assurer qu'elles correspondent bien aux tableaux successifs du programme considéré.

Une base de données très simple

Peut-être avez-vous déjà rencontré l'expression « base de données ». Elle désigne un ensemble ordonné d'informations que l'on peut passer en revue à l'aide de programmes conçus à cet effet, afin par exemple de demander à l'ordinateur d'en imprimer certains éléments. La plupart des systèmes informatisés de gestion utilisent des bases de données. Nous allons écrire un programme mettant en place une structure très simple de base de données, à l'aide des instructions READ et DATA. Nous pourrons consulter cette base de données, en indiquant à l'ordinateur les informations qui nous intéressent. Passons maintenant à l'application pratique : la constitution d'un catalogue de bibliothèque.

Nous aurons trois tableaux : un premier pour les numéros de référence, un deuxième pour les auteurs et un troisième pour les titres. Entrons les informations suivantes :

Réf.	*Auteur*	*Titre*
0001	DUMAS	VINGT ANS APRÈS
0002	HUGO	LES MISÉRABLES
0003	MELVILLE	MOBY DICK
0004	MOLIÈRE	L'AVARE
0005	TOURNIER	VENDREDI
0006	VERNE	LE RAYON VERT

Nous nommerons ces tableaux R, A$ et T$. Voici la sous-partie de programme ainsi obtenue :

```
100 DIM R(6):DIM A$(6):DIM T$(6)
110 FOR X = 1 TO 6
120 READ R(X):READ A$(X):READ T$(X)
130 NEXT X
140 DATA 0001,"DUMAS","VINGT ANS APRES"
150 DATA 0002,"HUGO","LES MISERABLES"
160 DATA 0003,"MELVILLE","MOBY DICK"
170 DATA 0004,"MOLIERE","L'AVARE"
180 DATA 0005,"TOURNIER","VENDREDI"
190 DATA 0006,"VERNE","LE RAYON VERT"
```

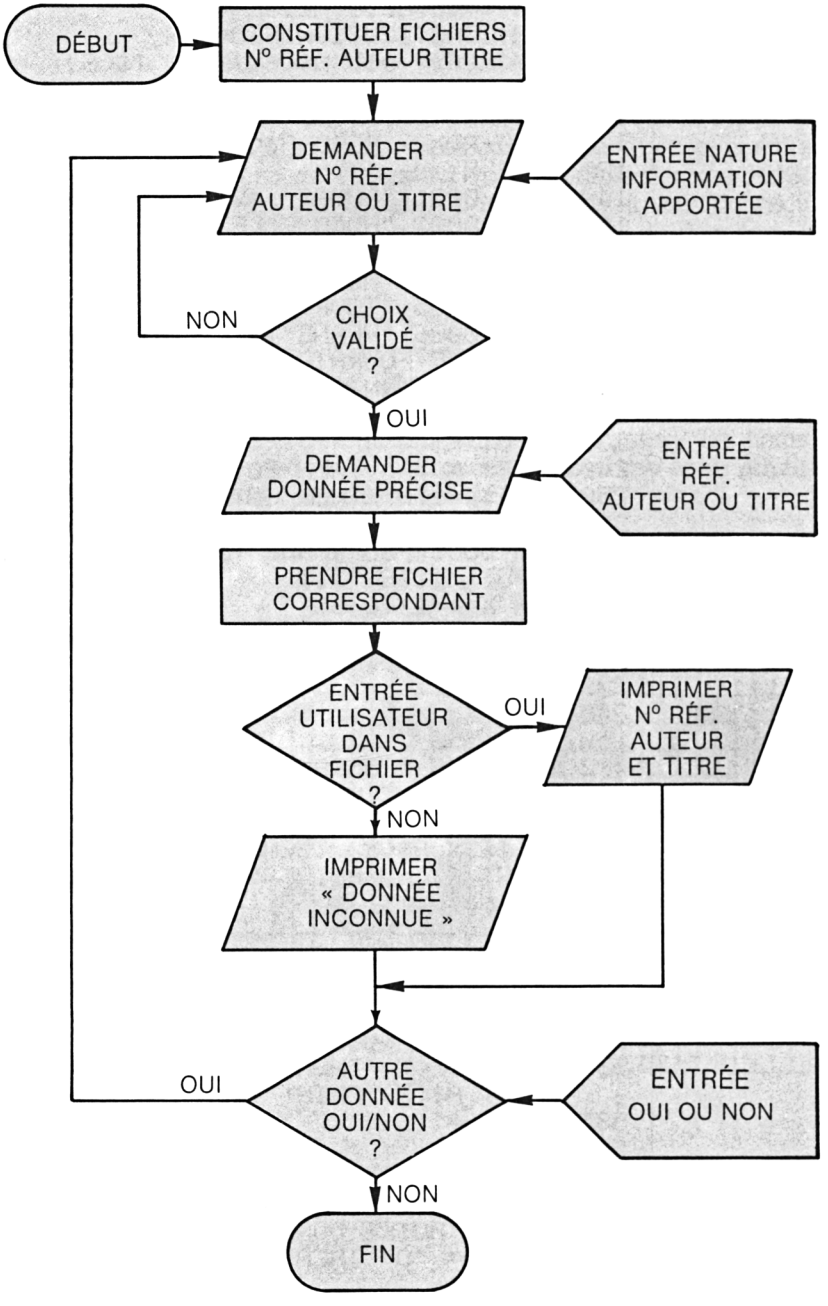
Remarquez avec quel soin nous avons ordonné nos instructions DATA afin que chaque article trouve sa bonne place. Qu'allons-nous faire de ces données? Nous pouvons par exemple demander à l'utilisateur de nous donner soit un numéro de référence, soit un nom d'auteur, soit un titre, et faire en sorte que l'ordinateur lui fournisse le reste des informations dont il dispose. L'organigramme vous montre comment l'on peut opérer. Encore une fois, il vous sera utile de le consulter en étudiant le programme.

Il faudra poser deux questions à l'utilisateur : 1° Quelle type d'information va-t-il nous fournir : numéro de référence, auteur ou titre? 2° Quelle est cette information, en clair? Nous utiliserons la variable F$ pour les réponses à la première question et les variables I$ (noms d'auteurs, titres) et I (variable numérique pour les numéros de référence) pour les réponses à la seconde. Nous utiliserons enfin les variables C et O$ pour respectivement contrôler si l'information a été trouvée ou non et savoir si l'utilisateur désire une autre information.

Voici donc la liste des variables :

Tableau des numéros de référence	R(1 à 6)
Tableau des noms d'auteurs	A$(1 à 6)
Tableau des titres	T$(1 à 6)
Information fournie par l'utilisateur	F$
Information de chaîne (auteurs, titres)	I$
Information numérique (n° de référence)	I
Variable de contrôle	C
Variable de ré-exécution	O$

Organigramme « bibliothèque »



Et voici notre programme :

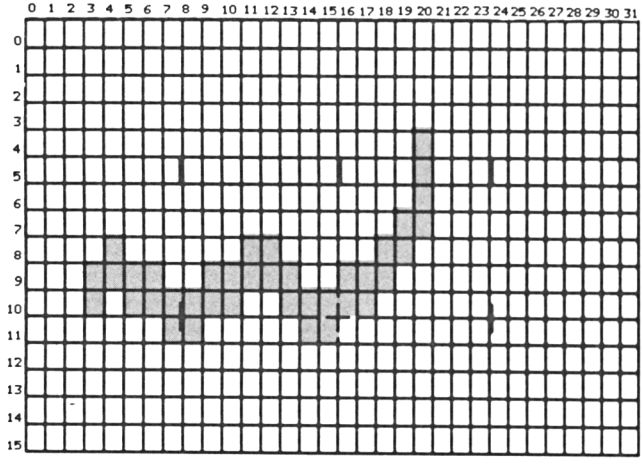
```
10  REM ***BIBLIOTHEQUE***
20  REM CONSTITUTION FICHIER
100 DIM R(6): DIM A$(6): DIM T$(6):
    LET C = 0
110 FOR X = 1 TO 6
120 READ R(X): READ A$(X): READ T$(X)
130 NEXT X
140 DATA 0001,"DUMAS","VINGT ANS APRES"
150 DATA 0002,"HUGO","LES MISERABLES"
160 DATA 0003,"MELVILLE","MOBY DICK"
170 DATA 0004,"MOLIERE","L'AVARE"
180 DATA 0005,"TOURNIER","VENDREDI"
190 DATA 0006,"VERNE","LE RAYON VERT"
200 REM INFO FOURNIE PAR UTILISATEUR
210 PRINT "FOURNISSEZ-VOUS NO.REF.,"
220 PRINT "AUTEUR OU TITRE (R,A OU T)"
230 INPUT F$
240 IF F$ = "R" THEN GOTO 500
250 IF F$ = "A" THEN GOTO 400
260 IF F$ <> "T" THEN GOTO 210
300 REM SOUS-PROGRAMME T
310 INPUT "TITRE";I$
320 FOR X = 1 TO 6
330 IF I$ = T$(X) THEN PRINT T$(X):
    PRINT A$(X):PRINT "REF.";R(X):
335 LET C = 1
340 NEXT X
350 GOTO 550
400 REM SOUS-PROGRAMME A
410 INPUT  "AUTEUR";I$
420 FOR X = 1 TO 6
430 IF I$ = A$(X) THEN PRINT A$(X) :
    PRINT T$(X):PRINT "REF.";R(X)
435 LET C = 1
440 NEXT X
450 GOTO 550
500 REM SOUS-PROGRAMME R
510 INPUT "NO.REF.(4 CHIFFRES)";I
520 FOR X = 1 TO 6
530 IF I = R(X) THEN PRINT R(X):
    PRINT A$(X):PRINT T$(X)
535 LET C = 1
540 NEXT X
550 REM SEQUENCE FINALE
560 IF C = 0 THEN PRINT"DONNEE INCONNUE"
570 INPUT "AUTRE DONNEE O/N";O$
580 IF O$ = "O" THEN LET C = 0:GOTO 200
590 END
```

Bien que ce programme soit assez long, il n'est néanmoins pas très complexe. Les vraies bases de données n'exigent pas, par exemple, une complète adéquation entre les entrées de l'utilisateur et les éléments d'information qu'elles contiennent. Notre programme, lui, ne fournira de réponse que si les articles de la base de données correspondent exactement aux entrées de l'utilisateur (par exemple l'entrée HUGOT ne sera pas acceptée). Néanmoins, il vous donne une bonne idée de ce que sont les bases de données.

Maintenant, c'est à vous

4) Écrivez un programme qui dessine un serpent se déroulant sur l'écran, en plaçant les données de position d'impression à l'intérieur d'instructions DATA.

5) Écrivez un programme qui contient les noms et adresses de 10 personnes, et imprime leurs adresses quand vous entrez leurs noms.



Serpent
Essayez de dessiner un serpent comme celui-ci. Il s'allonge bloc par bloc et ensuite sort sa langue venimeuse.

Contrôle

Arrivé à la fin de ce chapitre, vous devriez savoir :

Sur les tableaux

— comment dimensionner un tableau d'une colonne à l'aide de l'instruction DIM ;
— comment entrer des données dans un tableau en utilisant l'instruction INPUT ;
— comment placer des données dans un tableau en utilisant les instructions READ... DATA.

Sur les bases de données

— comment un ensemble ordonné de données peut jouer le rôle de base de données dans l'ordinateur ;
— comment constituer et consulter une base de données très simple en recourant à des instructions DATA et à des tableaux.

7

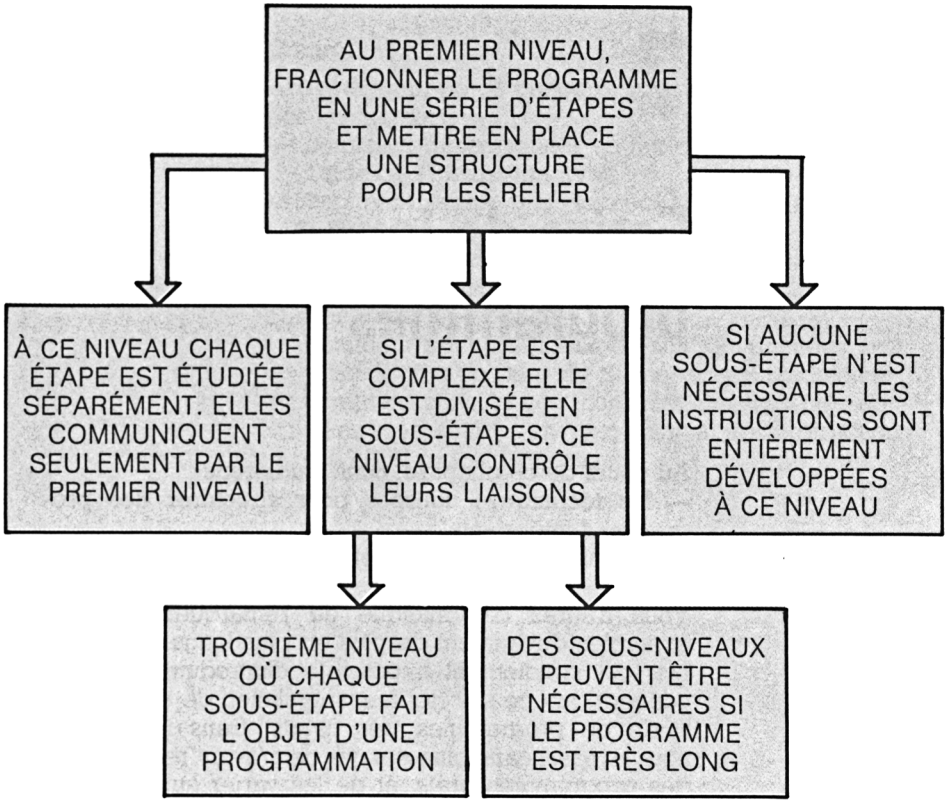
Écrire de longs programmes

Au cours de ce chapitre, nous étudierons :
— les techniques utilisées pour structurer des programmes importants,
— les instructions GOSUB et RETURN.

Vous n'aurez pas manqué de remarquer que nos exemples devenaient de plus en plus longs. C'est qu'il est tout simplement impossible d'introduire quelque concept que ce soit (tel la manipulation de données) à l'aide de programmes très courts. Dans l'ensemble, nous avons néanmoins essayé de n'avoir recours qu'à des programmes brefs, et de les varier au maximum. Nous adopterons dans ce chapitre l'approche inverse : nous avons choisi deux programmes longs, et travaillerons étape par étape à leur réalisation.

Structurer vos programmes

Vous avez peut-être déjà entendu parler de « programmation structurée ». Il s'agit d'une technique d'écriture des programmes qui met l'accent sur l'élaboration d'une structure logique, dans laquelle chaque instruction vient s'insérer. Le programme, dans un premier temps, est sommairement découpé en une série de sous-parties, ou modules, qui se relient de manière claire et bien définie. Chaque module est, si nécessaire, subdivisé en modules de plus en plus petits, jusqu'à ce que chacun d'entre eux se réduise à une taille qui le rende aisément maniable. Ensuite, les modules sont programmés séparément et font l'objet d'un contrôle aussi serré que possible, avant d'être assemblés. La programmation structurée est une manière sûre de réaliser de longs programmes, faciles à tester, à déboguer ou à modifier.

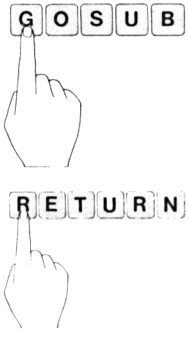


La programmation structurée
Voici une représentation schématique de cette démarche. Bien sûr, la structure d'un programme peut être beaucoup plus complexe.

Les programmeurs qui utilisent cette technique regardent avec méfiance la commande GOTO. Ils soutiennent que beaucoup tombent dans le travers de parsemer de longs listages d'instructions GOTO, sans nécessité apparente. Ils préfèrent des suites d'instructions plus rigoureuses pour les boucles et les branchements. Certaines d'entre elles n'existent pas en Basic, mais seulement dans des langages comme le Pascal. Quelques-unes (par exemple, la séquence IF... THEN... ELSE et les instructions de manipulation du BBC Basic) existent dans des versions Basic plus étoffées que celles utilisées par les ordinateurs familiaux.

Nous nous sommes attachés dans ce livre à vous familiariser avec les techniques d'écriture de programmes les plus simples. Nous avons utilisé l'instruction GOTO, par exemple, parce que nous pensons qu'elle est plus facile à comprendre et à manier que nombre d'instructions de structuration, et parce qu'on la trouve dans tous les Basic.

Les techniques que nous avons étudiées conviennent parfaitement aux programmes courts, mais, il faut l'admettre, elles deviennent insuffisantes lorsqu'il s'agit d'élaborer de longs programmes. Si vous décidez d'écrire de nombreux programmes longs, vous verrez rapidement à quel point il est facile de perdre le fil et combien il est préférable de disposer d'outils souples de structuration. A ce stade, vous serez « mûrs » pour entreprendre l'étude d'un livre plus ardu !



Dans ce chapitre, nous n'abordons pas la programmation structurée proprement dite, bien que nous y empruntions beaucoup de notions. Nous n'introduisons qu'une seule instruction nouvelle, très largement répandue : GOSUB... RETURN. Nous pensons que notre approche compense ses imperfections à cet égard, en rendant l'apprentissage plus facile pour le débutant. Toutefois, il est important de se souvenir qu'au fur et à mesure que la longueur des programmes augmente, les difficultés augmentent elles aussi, et cela plus que proportionnellement ! Les programmes importants requièrent un effort de logique accru et exigent qu'une attention toute particulière soit portée aux articulations de leurs diverses parties.

Programme papier peint

Notre premier programme portera sur une tâche de la vie réelle, beaucoup moins simple qu'il n'y paraît. Il s'agira de calculer la quantité de papier peint nécessaire pour tapisser une pièce.

Avant de penser aux instructions Basic que l'on va utiliser — et avant même de mettre en route l'ordinateur — il faut réfléchir globalement au programme : que va-t-on faire, et comment, dans les grandes lignes, va-t-on le faire ?

Il est possible de tapisser toutes sortes de surfaces, murs, plafonds. Il convient, premièrement, d'en déterminer la surface. Ce sera la première étape de notre plan de programmation. Il faut également connaître la quantité de papier peint contenue dans un rouleau ; ce sera la deuxième étape. En enfin, il faut savoir combien de rouleaux seront nécessaires au tapissage de toutes les surfaces ; ce sera la troisième étape.

Voici le plan général — le premier niveau, si vous préférez — du programme :

Étape 1 : calculer la surface à tapisser.
Étape 2 : calculer la surface couverte avec un rouleau de papier peint.
Étape 3 : calculer le nombre de rouleaux nécessaires.

Ces étapes forment trois parties séparées dans le programme. Nous les traiterons une par une ; mais, avant toute chose, il convient de s'intéresser à la façon de les assembler, à la cohérence du programme. Ainsi nous devrons veiller à bien employer la même unité de mesure (centimètres, mètres, etc.) dans toutes les parties, ou alors introduire une sous-partie destinée aux conversions. Les noms de variable devront également faire l'objet de toute notre attention. La cohérence d'un programme doit être établie dès le départ.

Dressons la liste des points importants :

L'unité de mesure. Peu importe l'unité elle-même, pourvu qu'elle ne varie pas au cours du programme. Prévoyons de placer une instruction PRINT dans l'introduction, pour rappeler à l'utilisateur d'être cohérent dans ses choix.

Les variables du programme. Ce sont les variables de base. Nous aurons sûrement besoin d'en introduire d'autres dans les niveaux secondaires. Nous utiliserons ces variables du début à la fin du programme ; les autres ne concerneront que les sous-programmes.

Surface à tapisser	S
Surface couverte par un rouleau	R
Nombre de rouleaux nécessaires	N

Il s'agit maintenant de commencer la transcription en Basic. Mais comment faire, direz-vous, sans savoir, dans le détail, de quoi seront composées les diverses parties et sans avoir donné de nom aux variables secondaires ? Tout simplement en prévoyant, dans un premier temps, d'utiliser des sous-programmes.

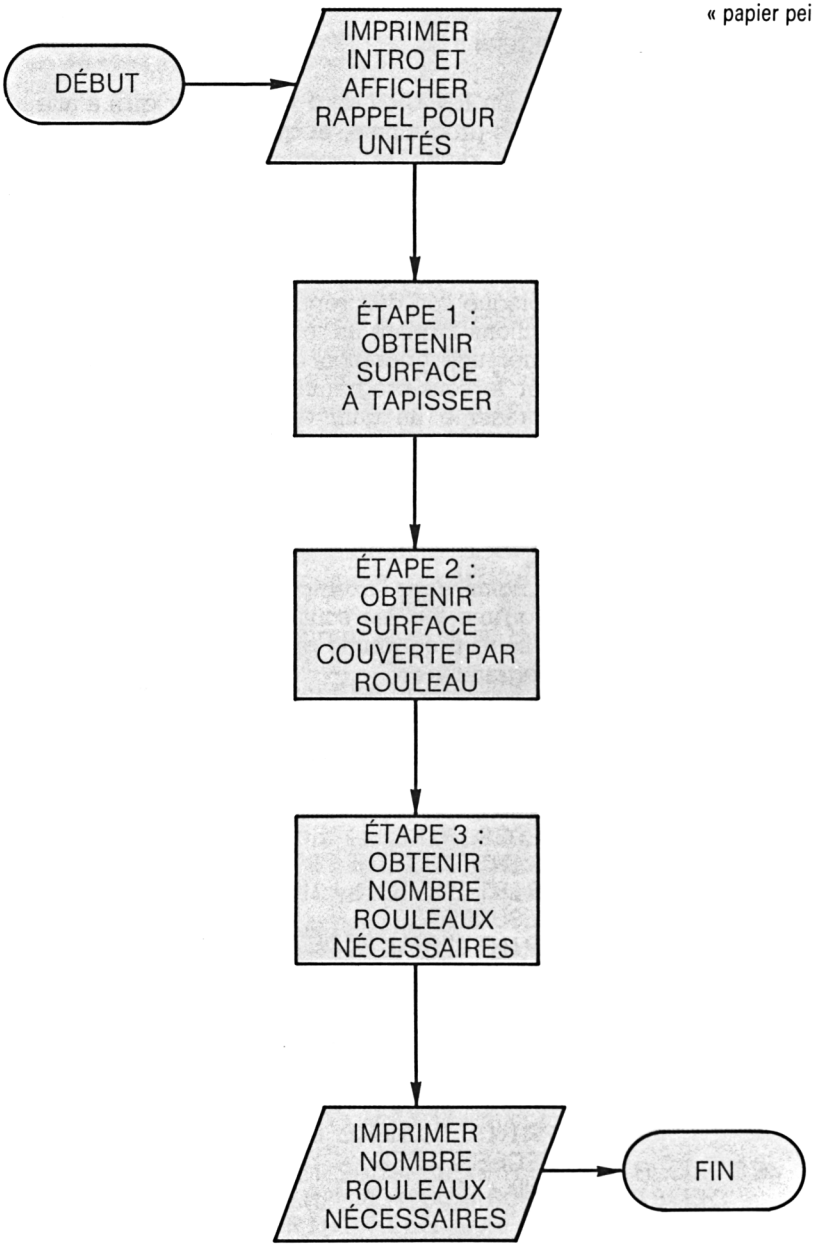
Un sous-programme est un programme à part entière, utilisé par le programme principal. Il peut servir plusieurs fois et comporte un ensemble spécifique de numéros de ligne (généralement plus grands que ceux utilisés dans le programme principal). Une instruction GOSUB permet au programme principal de l' « appeler ». C'est-à-dire que le programme principal donne l'ordre à la machine d'exécuter les instructions du sous-programme quand il en a besoin. Par exemple :

```
GOSUB 1000
```

veut dire : « Aller à la ligne 1000 et exécuter le sous-programme qui s'y trouve. » A partir de la ligne 1000, on trouvera une suite comme celle-ci :

```
1000 REM SOUS-PROGRAMME
```

Organigramme de 1er niveau « papier peint »



```
1010 ┐
...  } instructions composant le sous-programme
1060 ┘
1070 RETURN
```

L'instruction RETURN dit à l'ordinateur qu'il a atteint la fin du sous-programme, et qu'il doit revenir à l'endroit où il avait quitté le programme principal. RETURN fonctionne comme l'instruction GOTO ; mais l'ordinateur se rappelle ici de l'endroit où il doit revenir, aussi n'est-il pas besoin d'y adjoindre un numéro de ligne.

En fait, les sous-programmes sont particulièrement utiles lorsque l'on doit répéter plusieurs fois une série d'instructions formant un tout, permettant par exemple d'effectuer une opération algébrique précise. Il suffit de taper le sous-programme une fois pour toutes, ce qui représente un gain de place et une économie d'efforts. Peu importent le nombre de fois et l'endroit où l'on appelle le sous-programme ; l'ordinateur reviendra exactement là où il le faut dans le programme principal, après avoir exécuté les instructions du sous-programme.

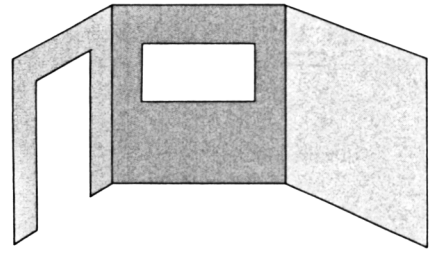
Néanmoins, dans l'exemple qui va suivre, nous n'utiliserons qu'une fois les sous-programmes. Voici l'ébauche de notre programme principal, faisant appel à trois sous-programmes :

```
10  REM *** PAPIER-PEINT ***
20  REM    PAR SUSAN CURRAN
30  PRINT "CE PROGRAMME CALCULE"
40  PRINT "LA QUANTITE DE PAPIER
    NECESSAIRE"
50  PRINT "AU TAPISSAGE DE LA PIECE"
60  PRINT "S.V.P, INDIQUEZ TOUTES LES
    MESURES"
70  PRINT "A L'AIDE DE LA MEME UNITE"
80  REM SURFACE A TAPISSER : S
90  REM SURFACE D'UN ROULEAU : R
100 REM NBRE DE ROULEAUX NECESSAIRES: N
110 GOSUB 1000 : REM OBTENIR S ⟹
120 GOSUB 2000 : REM OBTENIR R ⟹
130 GOSUB 3000 : REM OBTENIR N ⟹
140 PRINT "NOMBRE DE ROULEAUX
    NECESSAIRES : ";N
150 END
```

Tout ce qu'il nous reste à faire maintenant est d'élaborer nos trois sous-programmes de façon claire.

Surface à tapisser

Pour calculer la somme de papier peint nécessaire, on peut se représenter les murs et le plafond comme des panneaux rectangulaires dont certains seraient percés d'ouvertures elles-mêmes rectangulaires : les portes et les fenêtres.



Commençons par le sous-programme 1, celui qui calcule la surface à tapisser. Encore une fois, réfléchissons un instant à ce que nous allons tapisser. Les murs et les plafonds sont normalement rectangulaires, avec des ouvertures pour les portes, les fenêtres, etc. Afin de nous faciliter la tâche, nous nous en tiendrons à des panneaux percés d'ouvertures rectangulaires.

Nous tapisserons plus d'un mur avec le même papier ; il nous faut donc calculer plusieurs surfaces et les ajouter. Organisons notre sous-programme de la façon suivante :

a) Obtenir les dimensions d'un panneau.
b) En calculer la surface et l'ajouter au total déjà obtenu.
c) Retourner en a) s'il se trouve encore des panneaux à ajouter.
d) Obtenir les dimensions des ouvertures.
e) Calculer la surface des ouvertures et soustraire celle-ci du total.
f) Retourner en d) s'il se trouve encore des ouvertures à ajouter.

Nous aurons besoin des variables suivantes :

Premier paramètre de surface (hauteur)	H
Deuxième paramètre de surface (largeur)	L
Calcul total des surfaces	S

S, calculée par le sous-programme, est reprise par le programme principal. Les autres variables n'interviennent que dans le sous-programme.

Nous ne savons pas combien de surfaces ou d'ouvertures seront à prendre en compte ; aussi, nous utiliserons une boucle ouverte avec les instructions IF... THEN GOTO. Voici le sous-programme :

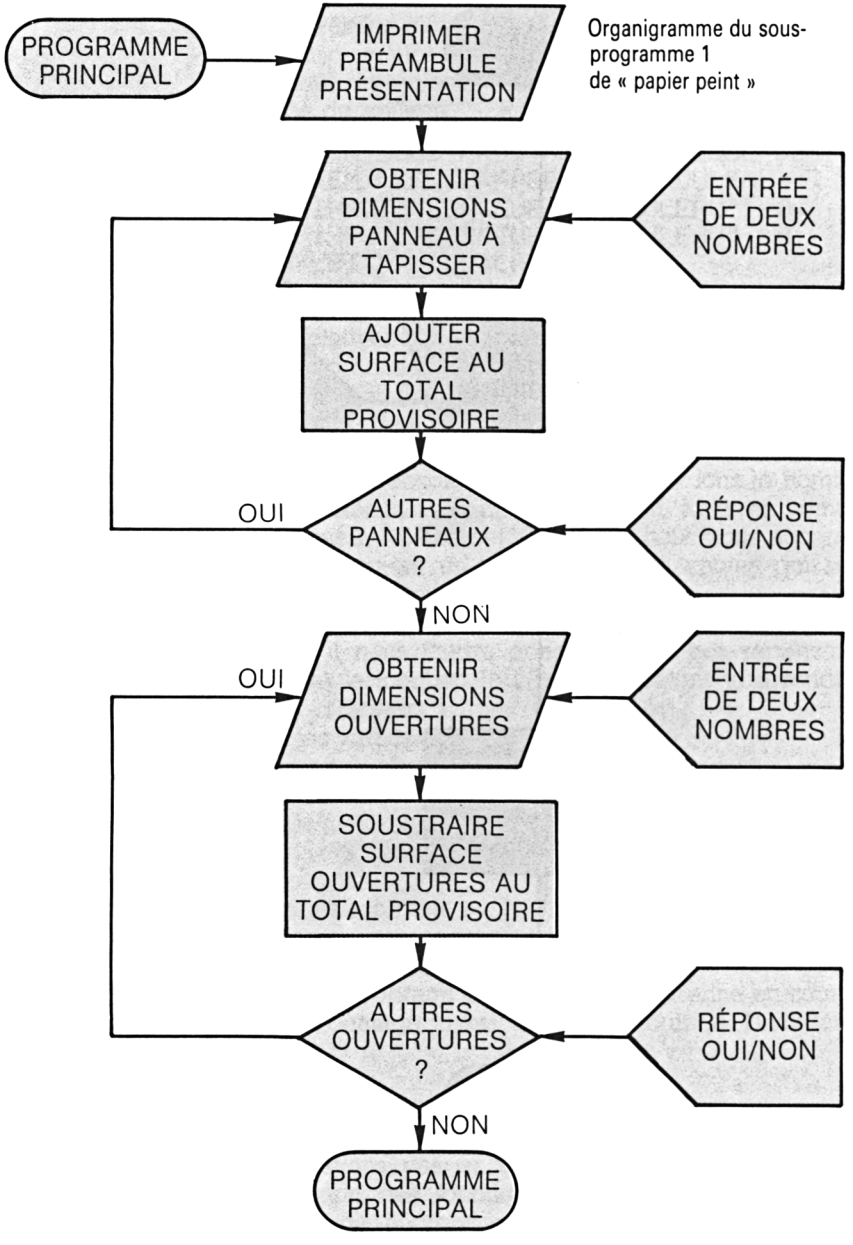
```
→1000 REM SOUS-PROGRAMME : CALCUL DE
 1010 REM   LA SURFACE A TAPISSER
 1020 PRINT "LES SURFACES A TAPISSER
      DOIVENT"
 1030 PRINT "ETRE CONSIDEREES COMME"
```

```
1040 PRINT "DES PANNEAUX RECTANGULAIRES"
1050 PRINT "QUI ONT DES OUVERTURES"
1060 PRINT "EGALEMENT RECTANGULAIRES."
1070 PRINT "ENTREZ LES DIMENSIONS
     DES PANNEAUX,"
1080 PRINT "PUIS CELLES DES OUVERTURES"
1090 LET S = 0
1100 INPUT "HAUTEUR DU PANNEAU";H
1110 INPUT "LARGEUR DU PANNEAU";L
1120 LET S = S+(H+L)
1130 INPUT "AUTRES PANNEAUX ? O/N";
     R$
1140 IF R$ = "O" THEN GOTO 1100
1150 INPUT "HAUTEUR DE L'OUVERTURE";H
1160 INPUT "LARGEUR DE L'OUVERTURE";L
1170 LET S = S - (H*L)
1180 INPUT "AUTRES OUVERTURES ? O/N";R$
1190 IF R$ = "O" THEN GOTO 1150
1200 RETURN
```

Tout cela est-il bien clair pour vous? Le sous-programme est suffisamment simple. Mais nous pouvons l'améliorer en le transformant en un sous-programme incluant deux sous-sous-programmes, le premier définissant les lignes générales et les seconds opérant les calculs sur les panneaux et les ouvertures. Cela ne pose aucune difficulté puisque l'on peut emboîter des sous-programmes de la même manière que l'on emboîte des boucles. Nous vous donnons le sous-programme et vous laissons l'initiative pour les deux « sous-sous-programmes » (vous en trouverez une version incluse dans le programme complet, pages 154-155).

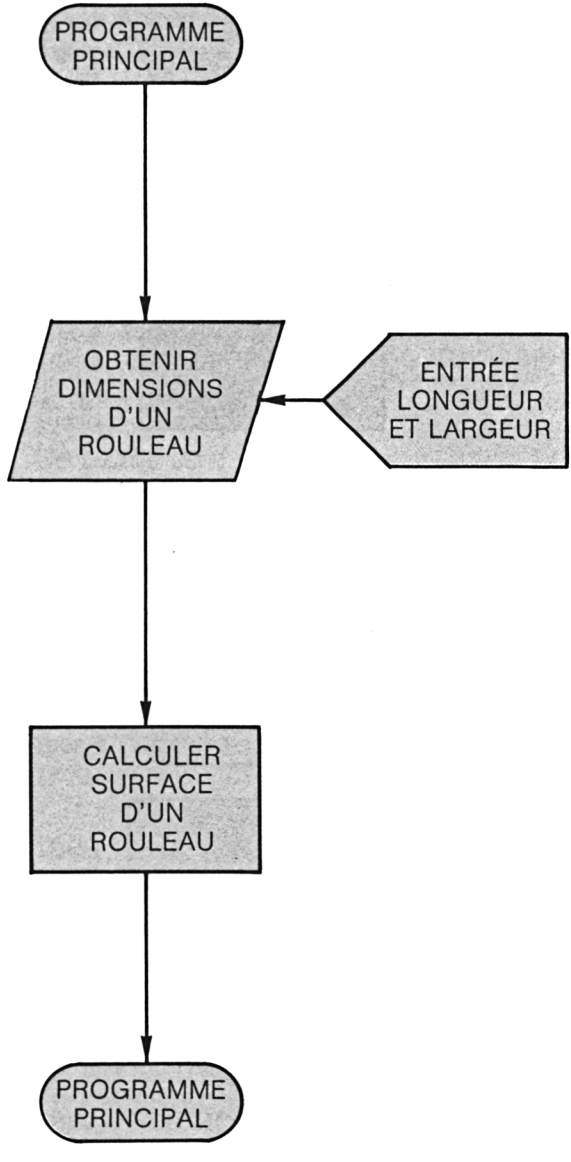
```
1000 REM SOUS-PROGRAMME : CALCUL DE
1010 REM   LA SURFACE A TAPISSER
1020 PRINT "LES SURFACES A TAPISSER
     DOIVENT"
1030 PRINT "ETRE CONSIDEREES COMME"
1040 PRINT "DES PANNEAUX RECTANGULAIRES"
1050 PRINT "QUI ONT DES OUVERTURES"
1060 PRINT "EGALEMENT RECTANGULAIRES."
1070 PRINT "ENTREZ LES DIMENSIONS
     DES PANNEAUX,"
1080 PRINT "PUIS CELLES DES OUVERTURES"
1090 LET S = 0
1100 GOSUB 15000 : REM CALCUL DU TOTAL
     DES SURFACES
```

```
1110 GOSUB 17000 : REM CALCUL DU TOTAL
     DIMINUE DES OUVERTURES ←
1120 RETURN
```



Organigramme du sous-programme 1 de « papier peint »

Organigramme du sous-programme 2 de « papier peint »



C'est clair ? Alors, passons maintenant au second sous-programme. Il sera plus court. Il suffit d'obtenir les dimensions d'un seul rouleau de papier et de calculer sa surface. Pas besoin de boucles. Nos variables seront à peu près les mêmes :

Largeur du rouleau	LA	} variables de sous-programme
Longueur du rouleau	LO	
Surface du rouleau	R	variable de programme

```
→2000 REM SOUS-PROGRAMME : CALCUL DE
 2010 REM  LA SURFACE D'UN ROULEAU
 2020 INPUT "LONGEUR DU ROULEAU";LO
 2030 INPUT "LARGEUR DU ROULEAU";LA
 2040 LET R = LO*LA
 2050 RETURN→
```

C'est relativement simple. Passons donc au troisième sous-programme. Cette fois-ci, il y a deux choses auxquelles il faut prêter attention. Premièrement, il convient de tenir compte des chutes de papier inutilisé (fins de rouleau, raccords). Si nous voulions être vraiment perfectionnistes, nous calculerions le nombre de lés obtenus avec chaque rouleau. Mais ne compliquons pas à l'excès cet exercice : nous nous accorderons simplement un pourcentage de chutes qui sera fixé par l'utilisateur.

De plus, nous ne pouvons acheter que des rouleaux entiers, il nous faudra donc arrondir nos réponses à l'aide de la fonction INT (qui, rappelons-le arrondit à l'entier inférieur).

Voici les variables :

Pourcentage de chutes	P	variable de sous-programme
Surface à tapisser	S	} variables du programme principal
Surface couverte par un rouleau	R	
Nombre de rouleaux	N	

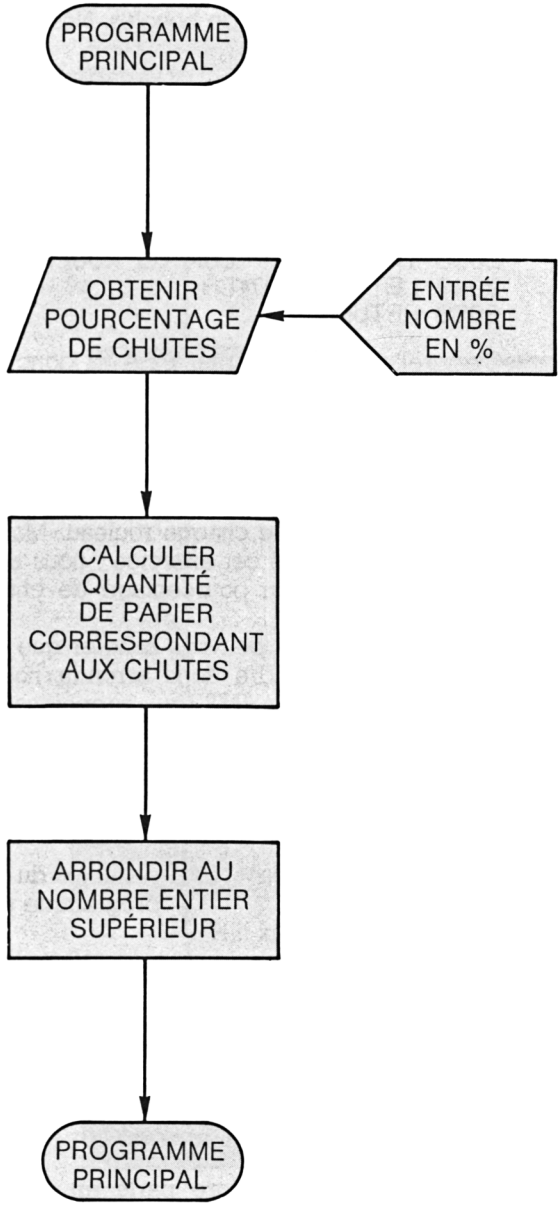
Pour un pourcentage de chute de 10 %, le calcul de la surface à tapisser sera : S × 110/100.

Aussi, afin d'obtenir une réponse qui prenne en compte ce pourcentage, nous aurons recours à l'instruction suivante :

```
LET N = (S*(P+100))/(R*100)
```

Nous n'avons pas encore parlé en détail de l'ordre d'exécution des fonctions arithmétiques ; nous avons introduit de nombreuses parenthèses dans notre ex-

Organigramme du sous-programme 3 de « papier peint »



pression pour être sûrs du résultat. Pourquoi ne pas la contrôler sur-le-champ? Si vous avez déjà entré une partie du programme principal, vous pouvez isoler votre contrôle en utilisant un numéro de ligne très petit, comme ceci :

```
1 LET S = 100: LET P = 10: LET R = 10
2 LET N = (S*(P+100))/(R*100)
3 PRINT N
4 END
```

Vérifiez le résultat obtenu, en effectuant vous-même le calcul (à l'aide, par exemple, d'une calculatrice). Voici le sous-programme complet :

```
→3000 REM SOUS-PROGRAMME DE CALCUL
 3010 REM DU NOMBRE DE ROULEAUX EMPLOYES
 3020 INPUT"POURCENTAGE DE CHUTES (%)";P
 3030 LET N = (S*(P+100)) / (R*100)
 3040 IF N >INT(N) THEN LET N = 1+INT(N)
 3050 RETURN→
```

Et voilà notre programme terminé. Élémentaire, en fin de compte, n'est-ce pas? Regardez le listage complet aux pages 154-155 et vous verrez que malgré sa relative longueur (51 lignes), le programme est très clair (et cela, grâce aux sous-programmes).

Il se peut que les organigrammes vous paraissent un peu superflus dans le cadre d'une approche structurée. Mais si, au contraire, vous tenez à les utiliser, il vous faudra en réaliser plusieurs, l'un pour définir le programme principal et les autres pour décrire les sous-programmes, comme nous l'avons fait aux pages précédentes.

Contrôle et mise au point

Tout n'est pas fini. Il faut encore nous assurer que le programme fonctionne correctement! faisons-le rapidement. Est-il possible de contrôler le programme sous-programme par sous-programme? Oui, si nous apportons quelques modifications afin d'isoler chaque sous-partie, un peu comme nous l'avions fait pour notre petit contrôle ci-dessus. Il nous faudra également vérifier que chaque sous-partie imprime un résultat correct. Pour ce faire, nous rajouterons un grand nombre d'instructions PRINT (ex. : PRINT S).

Il est bon de s'habituer à imprimer les résultats intermédiaires chaque fois que vous n'avez que des calculs simples à effectuer. De cette manière, vous

pouvez vérifier qu'ils sont plausibles. Exécutez le programme en utilisant des chiffres assez simples qui vous permettent d'effectuer les calculs vous-même et voyez si l'ordinateur obtient le même résultat. Il est important de prendre l'habitude de ce genre de contrôles afin d'être en mesure de détecter tout de suite les anomalies.

Programme « course de chevaux »

Nous allons nous divertir un peu en écrivant un programme de jeu simple, mais bien conçu, avec des séquences de titres et un marquage du score.

Il s'agira d'une course de chevaux se déroulant sur une piste (de la gauche vers la droite de l'écran) et mettant aux prises quatre chevaux. Nous inviterons le joueur à parier sur un cheval et nous le créditerons d'un avoir de 50 F. Ensuite nous suivrons l'évolution de sa bonne (ou mauvaise) fortune.

Au premier niveau, notre plan de programmation se présente ainsi :

Étape 1 : affichage de la séquence de titre et explication des règles du jeu.
Étape 2 : chaque joueur reçoit son avoir et effectue son pari.
Étape 3 : dessin de la piste.
Étape 4 : mouvement des chevaux sur la piste jusqu'à ce que l'un d'entre eux atteigne l'arrivée.
Étape 5 : résultat du pari et mise à jour de l'avoir.
Étape 6 : début d'une autre course, si le joueur le désire.
Étape 7 : fin du programme incluant les commentaires sur les résultats obtenus par le joueur.

De quelles variables avons-nous besoin à ce niveau du programme ? Premièrement, des variables représentant les sommes d'argent (les chevaux ayant tous une chance égale de gagner, nous nous en tiendrons à la cote de 4 contre 1).

Capital du joueur K
Mise du joueur M
Cheval sur lequel le joueur parie C

« C » est une variable numérique comme les autres puisque nous numérotons les chevaux de 1 à 4.

Donnons-leur également des noms et plaçons ces noms dans un tableau de chaîne.

C$(1) LUCKY LUKE
C$(2) GAI LURON
C$(3) ROQUEPINE
C$(4) CINQ DE MAI

Voici le programme dans ses grandes lignes, avec les sous-programmes des étapes principales :

```
10  REM *** COURSE DE CHEVAUX ***
20  LET K = 50 : REM NOM DES CHEVAUX
30  DIM C$(4) : REM  NOM DES CHEVAUX
40  FOR X = 1 TO 4: READ C$(X): NEXT X
50  DATA "LUCKY LUKE","GAI LURON"
60  DATA "ROQUEPINE","CINQ DE MAI"
70  GOSUB 1000 : REM TITRE
80  GOSUB 1500 : REM PARI
90  GOSUB 2000 : REM PLAN DE LA PISTE
100 GOSUB 2500 : REM LA COURSE
110 GOSUB 3000 : REM CONTROLE RESULTAT
120 IF K = 0 THEN GOTO 150
130 INPUT"VOULEZ-VOUS REJOUER (O/N)";R$
140 IF R$ = "O" THEN CLS : GOTO 80

150 GOTO 3500 : REM SEQUENCE FINALE
```

Cela vous semble sans doute un peu abstrait. Mais considérez cette première ébauche comme un index renvoyant aux différentes parties du programme. Nous serons tenus à plus de rigueur lorsque nous devrons écrire nos sous-programmes. Remarquez que la séquence finale ne sera pas à proprement parler un sous-programme, puisque nous n'effectuerons alors pas de retour (RETURN).

Mettons en place les titres et l'explication des règles. Que diriez-vous d'un titre clignotant ? Nous effacerons l'écran et les mots « courses sur ordinateur » clignoteront pendant quelques secondes. Vous souvenez-vous comment obtenir ce résultat ? On alterne l'effacement de l'écran et l'inscription du message en utilisant une boucle FOR... NEXT. Lorsque les tours de boucle s'achèveront, nous nous assurerons que l'écran est effacé et nous expliquerons les règles, ce qui nous

amènera à l'endroit du programme où seront éventuellement lancées de nouvelles courses. Là, nous devrons attendre un instant, avant d'effacer l'écran et de revenir au programme principal.
Ce sous-programme se divise en deux parties :

```
1000 REM SEQUENCE TITRE
1010 CLS
1020 GOSUB 1100 : REM TITRE CLIGNOTANT⇄
1030 GOSUB 1200 : REM REGLES DU JEU ⇄
1040 RETURN ⟶
1100 REM TITRE CLIGNOTANT
1110 FOR N = 0 TO 200
1120 PRINT@ (8,9),"COURSES SUR
     ORDINATEUR"
1130 CLS
1140 NEXT N
1150 RETURN ⟶
1200 REM REGLES
1210 PRINT"VOUS AVEZ 50F POUR PARIER"
1220 PRINT"MISEZ CE QUE VOUS DESIREZ"
1230 PRINT"SUR LE CHEVAL DE VOTRE CHOIX"
1240 PRINT"PARIEZ AUTANT DE FOIS QUE"
1250 PRINT"VOUS LE DESIREZ ,MAIS MISEZ"
1260 PRINT"QUELQUE CHOSE A CHAQUE FOIS"
↵1270 FOR D = 0 TO 5000 : NEXT D
1280 CLS
1290 RETURN ⟶
```

N'hésitez pas à modifier le texte là où bon vous semble. Essayez différentes valeurs pour la boucle de temporisation, jusqu'à ce que vous trouviez celle qui vous convient.

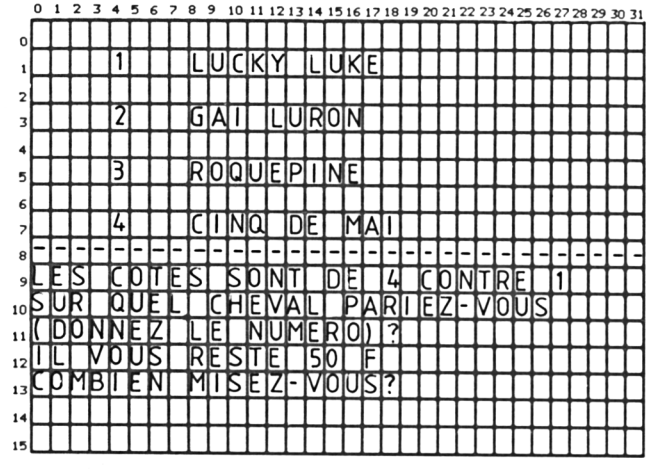
Notre sous-programme exécuté, nous nous retrouvons devant un écran vide, aussi, passons au sous-programme des paris. Nous y introduirons les quatre chevaux et nous demanderons au joueur sur quel cheval il parie et quel est le montant de sa mise. L'illustration de la page suivante vous montre la disposition que nous avons choisie. Voici le sous-programme qui permet de l'obtenir :

```
→1500 REM SOUS-PROGRAMME : PARI
 1510 REM IMPRESSION DES NOMS DE CHEVAUX
 1520 FOR X = 1 TO 4
 1530 PRINT: PRINT "     ";X;"     ";C$(X)
 1540 NEXT X
 1550 FOR X = 0 TO 31: PRINT "_";:NEXT X
```

```
1560 PRINT "LES COTES SONT
     DE 4 CONTRE 1"
1570 PRINT "SUR QUEL CHEVAL
     PARIEZ-VOUS ?"
1580 INPUT "(DONNEZ LE NUMERO)";C
1590 PRINT "IL VOUS RESTE ";K;"F"
1600 INPUT "COMBIEN MISEZ-VOUS";M
1610 IF M > K THEN GOTO 1600
1620 LET K = K - M
1630 CLS
1640 RETURN ⟶
```

Les paris

Voici la disposition pour laquelle nous avons optée dans la sous-partie des paris. On rappelle au joueur la somme dont il dispose, et il entre le numéro du cheval qu'il a choisi et le montant de sa mise.



Remarquez que nous utilisons des instructions PRINT vides pour espacer les noms, et notez la façon dont nous utilisons les variables de programme C, K, M.

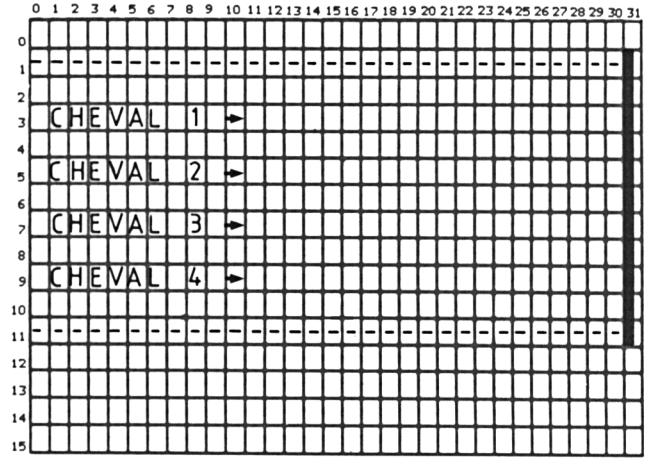
Passons maintenant à la partie suivante : le dessin de la piste. Encore une fois, ayons recours au crayon et au papier pour mettre en page, à l'avance, ce qui devra apparaître à l'écran. Il nous faudra déterminer les rangées sur lesquelles courront les chevaux (que nous ne dessinerons qu'ensuite). La piste restera très sommaire. Notre version n'a rien d'élaboré, mais vous pouvez l'améliorer, si bon vous semble, en dessinant des spectateurs, des arbres, etc.

En voici le sous-programme :

```
2000 REM DESSIN DE LA PISTE
2010 FOR X = 0 TO 31 :PRINT@ (1,X),"-";
     :NEXT X
2020 FOR X = 0 TO 31 :PRINT(11,X),"-";
     :NEXT X
```

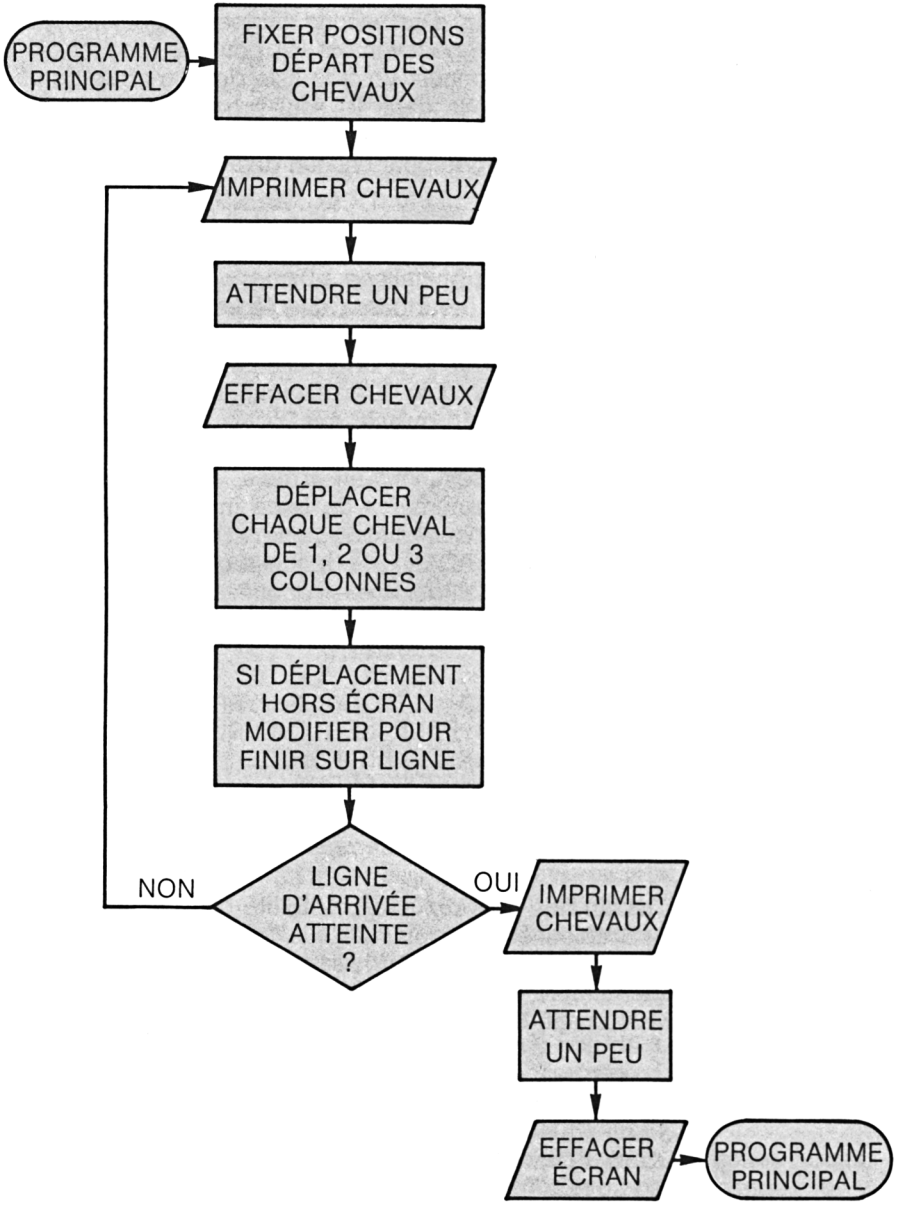
```
2030 FOR F = 1 TO 11 :
     PRINT@(F,31),CHR$(133);:NEXT F
2040 RETURN
```

La piste de course
Voici notre version de la piste. Chaque cheval a son couloir. La ligne d'arrivée se trouve à droite de l'écran, elle est composée de 11 carrés graphiques.



Nous abordons enfin le cœur du programme : la course elle-même ! Plusieurs choses sont à considérer. D'abord, nous ne pouvons effacer l'écran à chaque déplacement des chevaux, à moins de redessiner continuellement la piste, ce qui a l'inconvénient de provoquer un affichage trop saccadé. Aussi, nous attendrons un court instant après chaque série de déplacements, et nous effacerons les chevaux, en imprimant un vide à la place de chacun d'eux ; nous calculerons ensuite leur nouvelle position et puis nous les redessinerons. Ensuite, il nous faudra vérifier si l'un des chevaux n'a pas gagné la course ! Il suffira simplement de s'assurer que son numéro de colonne ne dépasse pas 31 (sachant qu'un cheval n'occupe qu'une colonne d'impression). S'il le dépasse, nous le ramènerons à 31 afin de ne pas sortir de la surface de l'écran et nous affecterons la valeur 1 à la variable G (pour « gagné »), que nous aurons mise à zéro au départ de chaque course. Ensuite, nous effectuerons un branchement sur un autre sous-programme qui imprimera le dernier déplacement de la course et retournera (RETURN) au programme principal. Bien sûr, il nous faut contrôler tous les chevaux pour déceler les ex-æquos. Si vous êtes un peu perdu, jetez un coup d'œil à l'organigramme décrivant cette séquence.

Organigramme du sous-programme « course »



Maintenant, les variables. Les rangées sur lesquelles nous allons imprimer les chevaux ne varieront pas, mais il nous faut tout de même les placer dans un tableau. Les positions de colonne, elles, varieront en fonction des déplacements des chevaux sur l'écran. Nous aurons donc besoin d'un second tableau. Appelons les deux tableaux respectivement RA pour les rangées, et CO pour les colonnes. Voici la liste des variables dont nous nous servons dans le sous-programme « course » :

Positions de rangée pour chacun des chevaux	RA (1 à 4)
Positions de colonne pour chacun des chevaux	CO (1 à 4)
Variable du gain (initialisée à 0)	G

Ici, nous nous heurtons à une légère difficulté que nous n'avions pas encore rencontrée. Il est impossible d'inclure les instructions DIM dans un sous-programme appelé plusieurs fois. L'ordinateur ne dimensionne (DIM) un tableau qu'une fois pour toutes. Si nous répétons cette opération, nous obtiendrons un signal d'erreur. Aussi, revenons en arrière, et utilisons quelques-uns de ces numéros de ligne inutilisés, qui sont si commodes, pour dimensionner (DIM) nos deux tableaux et lire (READ) les données du tableau RA. Il ne faut pas, à ce stade, lire les données du tableau CO, puisque la valeur de CO est remise à zéro à chaque fois que l'on relance une course.

```
 65  DIM RA(4): DIM CO(4): REM RANGEES /
     COLONNES POUR LES CHEVAUX
 66  REM CHOIX DE LA RANGEE
 67  REM DE CHAQUE CHEVAL
↱68  FOR X = 1 TO 4: READ RA(X): NEXT X
 69  DATA 3,5,7,9
```

Un cheval

Chacun des chevaux a la possibilité d'avancer de 1, 2 ou 3 colonnes à la fois ; c'est la fonction RND qui en détermine le nombre au hasard. Un caractère graphique simple fera office de cheval (voir illustration). Maintenant, écrivons notre sous-programme :

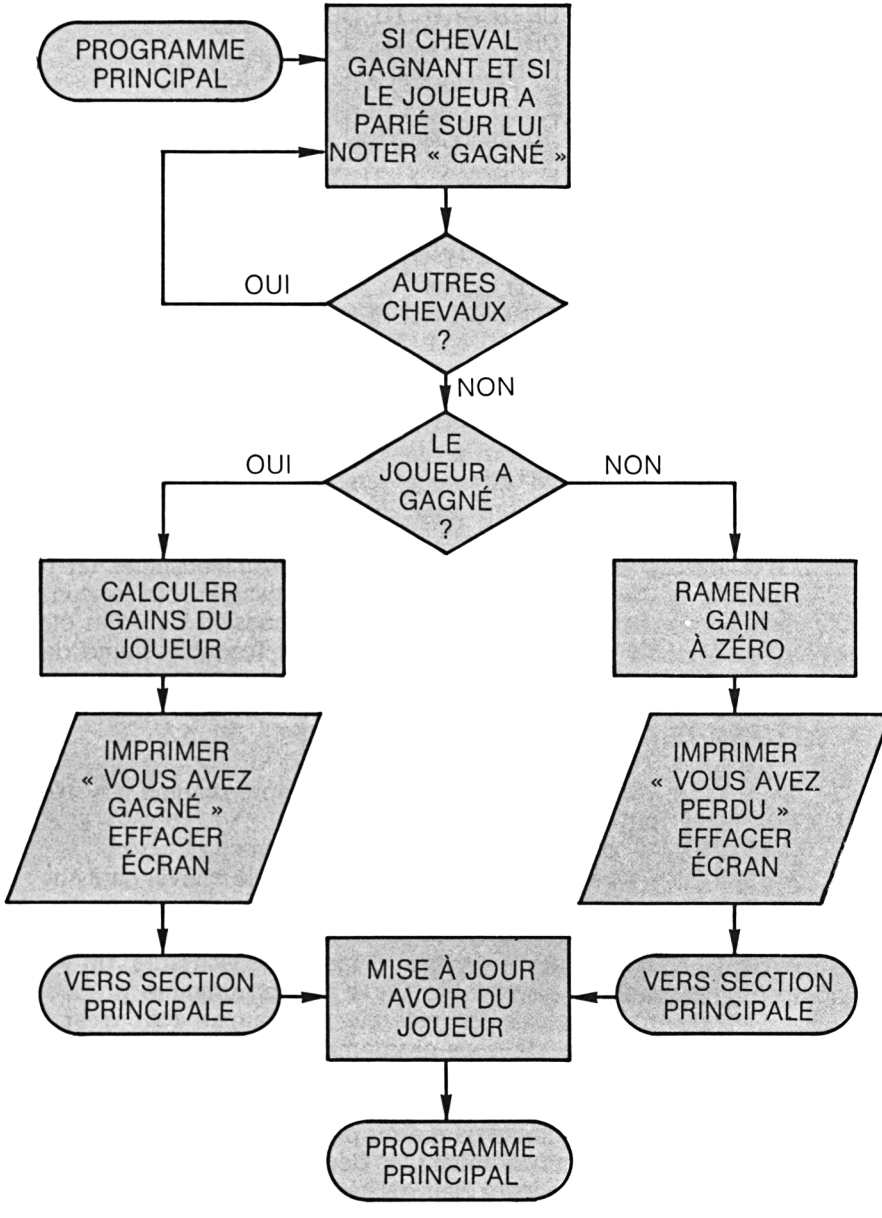
```
→2500 REM COURSE
 2510 LET G = 0: REM VARIABLE DU GAIN
 2520 REM POSITIONS COLONNE DE DEPART
↱2530 FOR X = 1 TO 4 : LET CO(X) = 0 :
      NEXT X
```

```
2540 REM BOUCLE DE LA COURSE
2550 FOR X = 1 TO 4
2560 PRINT@ ( RA(X),CO(X) ),CHR$(137);
2570 NEXT X
2580 FOR D = 0 TO 50 : NEXT D
2590 FOR X = 1 TO 4
2600 PRINT@ ( RA(X),CO(X) )," ";
2610 NEXT X
2620 FOR X = 1 TO 4
2630 LET CO(X) = CO(X) + 1 + INT(RND*3)
2640 IF CO(X) > 31 THEN LET CO(X) = 31:
     LET G = 1
2650 NEXT X
2660 IF G = 1 THEN GOSUB 2800: RETURN
2670 GOTO 2540
2800 REM DEPLACEMENT GAGNANT
2810 FOR X = 1 TO 4
2820 PRINT@ (RA(X),CO(X)),CHR$(137);
2830 NEXT X
2840 FOR D = 0 TO 500: NEXT D
2850 CLS
2860 RETURN
```

Il est suffisamment complexe pour constituer un programme à part entière, n'est-ce pas ? Si vous rencontrez des difficultés dans la compréhension de cette étape de la programmation, consultez l'organigramme de la page 147 ; il pourra vous aider à bien voir quelle tâche du sous-programme chaque groupe d'instructions accomplit.

Passons maintenant au sous-programme : « résultat ». Il nous faut :
— déterminer quel cheval a gagné,
— vérifier si le joueur a misé sur le cheval gagnant,
— informer le joueur de ses gains ou de ses pertes,
— mettre à jour l'avoir du joueur.
Nous aurons besoin d'une autre variable de gain. Comme la variable G du sous-programme précédent n'affecte pas le reste du programme (car c'est une variable de sous-programme), nous pouvons encore utiliser G. Mais n'oublions pas de la remettre à zéro ! Afin de pouvoir utiliser des messages différents selon que le joueur gagne ou perd, incorporons deux sous-programmes traitant les deux possibilités.
Est-ce toujours clair ? N'hésitez pas à consulter l'organigramme « résultat », si vous avez besoin d'aide.

Organigramme du sous-programme « résultat »



```
→3000 REM SOUS-PROGRAMME : RESULTAT
 3010 REM LE JOUEUR A-T-IL GAGNE
 3020 LET G = 0 : REM VARIABLE DU GAIN
┌→3030 FOR X = 1 TO 4
│ 3040 IF CO(X) = 31 THEN IF C = X
│      THEN LET G = 1
└─3050 NEXT X
 3060 IF G = 1 THEN GOSUB 3300 ⟷
 3070 IF G = 0 THEN GOSUB 3400 ⟷
 3080 LET K = K + M
 3090 RETURN ⟶
→3300 REM SOUS-PROGRAMME: JOUEUR GAGNANT
 3310 LET M = M*4
 3320 PRINT@ (6,10),"FELICITATIONS"
 3330 PRINT C$(C);" A GAGNE LA COURSE"
 3340 PRINT "VOUS AVEZ GAGNE ";M;"F"
 3350 FOR D = 0 TO 1000 : NEXT D
 3360 CLS
 3370 RETURN ⟶
→3400 REM SOUS-PROGRAMME: JOUEUR PERDANT
 3410 LET M = 0
 3420 PRINT@ (6,10),"PAS DE CHANCE"
 3430 PRINT C$(C);" N'A PAS GAGNE"
 3440 FOR D = 0 TO 1000 : NEXT D
 3450 CLS
 3460 RETURN ⟶
```

C'est presque fini! Il ne nous reste plus qu'à écrire la séquence finale. Examinons les résultats obtenus par le joueur et concluons par un message approprié.

```
→3500 REM SEQUENCE FINALE
 3510 IF K = 0 THEN GOTO 3600 ⟶
 3520 IF K < 50 THEN GOTO 3700 ⟶
 3530 IF K > 50 THEN GOTO 3800 ⟶
→3600 REM LE JOUEUR N'A PLUS D'ARGENT
 3610 PRINT@ (6,10),"VOUS N'AVEZ PLUS
      D'ARGENT"
 3620 PRINT "VOUS AUREZ PLUS DE CHANCE"
 3630 PRINT "LA PROCHAINE FOIS"
 3640 END
→3700 REM LE JOUEUR A PERDU, MAIS
      IL LUI RESTE DE L'ARGENT
 3710 PRINT@ (6,10),"PAS FAMEUX"
 3720 PRINT "VOTRE CAPITAL EST DE";K;"F"
 3730 PRINT "VOUS AUREZ PLUS DE CHANCE"
 3740 PRINT "LA PROCHAINE FOIS"
 3750 END
→3800 REM LE JOUEUR A GAGNE DE L'ARGENT
```

```
3810 PRINT@ (6,10),"BIEN JOUE"
3820 PRINT "VOUS ARRETEZ ALORS QUE"
3830 PRINT "VOUS GAGNEZ"
3840 PRINT "VOUS POSSEDEZ MAINTENANT "
     K;"F"
3850 END
```

Le programme est listé dans son ensemble aux pages 155-158. Examinez-le bien pour voir à quoi il ressemble, une fois écrit dans sa totalité. Et n'oubliez pas de le tester consciencieusement après l'avoir entré dans votre ordinateur.

Réduire la taille d'un programme

Concluons ce chapitre sur un aspect qui se révélera très important lorsque vous écrirez des programmes plus longs. Quelle est la taille de votre ordinateur? La véritable réponse réside dans la capacité de sa mémoire à accès direct (MEV). C'est elle, en effet, qui détermine l'espace disponible pour mémoriser votre programme et en réaliser l'exécution. Les plus petits micro-ordinateurs domestiques disposent seulement d'1 Ko (un Ko, ou kilo octet, représente 1.024 octets). Les plus grands disposent de 64 Ko.

Une mémoire vive (MEV) de faible capacité se sature rapidement. Par exemple, les programmes de ce chapitre n'entreront pas dans une mémoire limitée à 1 Ko. Quelle taille nécessitent-ils? Cela dépend, dans une certaine mesure, de la conception de la mémoire de votre ordinateur et de la façon dont vous tapez votre programme. Nous les avons testés en limitant la mémoire disponible de votre ordinateur — qui est de 16 Ko — à 3,5 Ko. Nous n'avons rencontré aucune difficulté.

Si vous vous trouvez à court d'espace mémoire alors que vous tapez ou exécutez votre programme, ce message vous le signalera : « OUT OF MEMORY » (mémoire saturée). Vous pouvez alors éliminer toutes les parties du programme qui occupent de l'espace mémoire inutile. Voici quelques suggestions :

1) Sur la plupart des ordinateurs, un blanc tapé occupe de l'espace mémoire, même si l'ordinateur l'ignore lorsqu'il décode l'instruction BASIC. Éliminez autant de blancs que l'ordinateur vous le permettra (il faudra en supprimer un grand nombre, peut-être tous ceux du listage).

2) Les REMarques prennent de la place. Nous en avons fait le principe de base de l'organisation de nos programmes parce qu'elles sont faciles à comprendre et que, grâce à elles, les notes explicatives restent toujours présentes dans le programme. Cependant, si vous manquez de place, supprimez les instructions REM. Conservez ces remarques — ou toute autre complément d'information relatif au fonctionnement de votre programme — sur papier, afin de pouvoir vous y référer, au besoin.

3) Les numéros de ligne occupent également de l'espace. Si votre ordinateur vous le permet, placez plusieurs instructions sur chaque ligne, dès que cela s'avère possible, et sans que la structure du programme en soit modifiée. La présentation de ce dernier, lorsque vous le listerez, sera moins claire, mais il ne s'en exécutera pas moins exactement de la même façon.

Vous serez bientôt en mesure d'estimer la taille du programme que votre ordinateur peut accepter. Dans le cas où sa mémoire vive dépasse 16 Ko, vous n'aurez probablement jamais à vous préoccuper de l'espace mémoire au cours de l'élaboration de vos propres programmes. La question ne se posera que pour ceux que vous vous procurerez tout faits. Si, malgré toutes ces mesures draconiennes, le programme n'entre pas dans la mémoire de l'ordinateur, il vous faudra alors, soit réduire vos ambitions, soit vous procurer une mémoire auxiliaire, soit songer à acheter une machine plus importante.

Listages des programmes

Vous trouverez ci-dessous les listages complets des deux programmes que nous avons réalisés au cours de ce chapitre.

Le programme « Papier peint »

Variables utilisées :	
Surface à recouvrir	S
Surface d'un rouleau	R
Nombre de rouleaux nécessaires	N
Variables de sous-programme :	
Dimensions des panneaux/ouvertures	H,L
Dimensions d'un rouleau	LO,LA
Pourcentage de chutes	P(P%)
Réponses aux questions OUI/NON	R$

```
10  REM *** PAPIER-PEINT ***
20  REM   PAR SUSAN CURRAN
30  PRINT "CE PROGRAMME CALCULE"
40  PRINT "LA QUANTITE DE PAPIER
    NECESSAIRE"
50  PRINT "AU TAPISSAGE DE LA PIECE"
60  PRINT "S.V.P. INDIQUEZ TOUTES LES
    MESURES"
70  PRINT "A L'AIDE DE LA MEME UNITE"
80  REM SURFACE A TAPISSER : S
90  REM SURFACE D'UN ROULEAU : R
100 REM NBRE DE ROULEAUX NECESSAIRES: N
110 GOSUB 1000 : REM OBTENIR S
120 GOSUB 2000 : REM OBTENIR R
130 GOSUB 3000 : REM OBTENIR N
140 PRINT "NOMBRE DE ROULEAUX
    NECESSAIRES : ";N
150 END
1000 REM SOUS-PROGRAMME : CALCUL DE
1010 REM   LA SURFACE A TAPISSER
1020 PRINT "LES SURFACES A TAPISSER
     DOIVENT"
1030 PRINT "ETRE CONSIDEREES COMME"
1040 PRINT "DES PANNEAUX RECTANGULAIRES"
1050 PRINT "QUI ONT DES OUVERTURES"
1060 PRINT "EGALEMENT RECTANGULAIRES."
1070 PRINT "ENTREZ LES DIMENSIONS
     DES PANNEAUX,"
1080 PRINT "PUIS CELLES DES OUVERTURES"
1090 LET S = 0
1100 GOSUB 15000 : REM CALCUL DU TOTAL
     DES SURFACES
1110 GOSUB 17000 : REM CALCUL DU TOTAL
     DIMINUE DES OUVERTURES
1120 RETURN
1500 REM SOUS-PROGRAMME :  TOTAL
     DES SURFACES
1510 INPUT "HAUTEUR DU PANNEAU";H
1520 INPUT "LARGEUR DU PANNEAU";L
1530 LET S = S + (H*L)
1540 INPUT "AUTRES PANNEAUX (O/N)";R$
1550 IF R$ = "O" THEN GOTO 1510
1560 RETURN
1700 REM SOUS-PROGRAMME : TOTAL DIMINUE
     DES OUVERTURES
1710 INPUT "HAUTEUR DE L'OUVERTURE";H
1720 INPUT "LARGEUR DE L'OUVERTURE";L
1730 LET S = S - (H*L)
1740 INPUT "AUTRES OUVERTURES (O/N)";R$
```

```
1750 IF R$ = "O" THEN GOTO 1710
1760 RETURN
2000 REM SOUS-PROGRAMME : CALCUL DE
2010 REM  LA SURFACE D'UN ROULEAU
2020 INPUT "LONGEUR DU ROULEAU";LO
2030 INPUT "LARGEUR DU ROULEAU";LA
2040 LET R = LO*LA
2050 RETURN
3000 REM SOUS-PROGRAMME DE CALCUL
3010 REM DU NOMBRE DE ROULEAUX EMPLOYES
3020 INPUT"POURCENTAGE DE CHUTES (%)";P
3030 LET N = (S*(P+100)) / (R*100)
3040 IF N >INT(N) THEN LET N = 1+INT(N)
3050 RETURN
```

Le programme : « Course de chevaux »

Variables utilisées :

Capital du joueur	K
Mise du joueur	M
Cheval choisi	C
Noms des chevaux	C$ (1 à 4)
Rangées de position des chevaux	RA (1 à 4)
Colonnes de position des chevaux	CO (1 à 4)
Variable du gain (sous-programmes)	G
Compteurs de boucles X,N,D(boucle de temporisation)	
Choix pour rejouer une nouvelle partie	R$

```
10  REM *** COURSE DE CHEVAUX ***
20  LET K = 50 : REM NOM DES CHEVAUX
30  DIM C$(4) : REM  NOM DES CHEVAUX
40  FOR X = 1 TO 4: READ C$(X): NEXT X
50  DATA "LUCKY LUKE","GAI LURON"
60  DATA "ROQUEPINE","CINQ DE MAI"
65  DIM RA(4): DIM CO(4): REM RANGEES /
    COLONNES POUR LES CHEVAUX
66  REM CHOIX DE LA RANGEE
67  REM DE CHAQUE CHEVAL
68  FOR X = 1 TO 4: READ RA(X): NEXT X
69  DATA 3,5,7,9
70  GOSUB 1000 : REM TITRE
80  GOSUB 1500 : REM PARI
90  GOSUB 2000 : REM PLAN DE LA PISTE
100 GOSUB 2500 : REM LA COURSE
110 GOSUB 3000 : REM CONTROLE RESULTAT
120 IF K = 0 THEN GOTO 150
130 INPUT"VOULEZ-VOUS REJOUER (O/N)";R$
140 IF R$ = "O" THEN CLS : GOTO 80
150 GOTO 3500 : REM SEQUENCE FINALE
1000 REM SEQUENCE TITRE
1010 CLS
```

```
1020 GOSUB 1100 : REM TITRE CLIGNOTANT
1030 GOSUB 1200 : REM REGLES DU JEU
1040 RETURN
1100 REM TITRE CLIGNOTANT
1110 FOR N = 0 TO 200
1120 PRINT@ (8,9),"COURSES SUR
     ORDINATEUR"
1130 CLS
1140 NEXT N
1150 RETURN
1200 REM REGLES
1210 PRINT"VOUS AVEZ 50F POUR PARIER"
1220 PRINT"MISEZ CE QUE VOUS DESIREZ"
1230 PRINT"SUR LE CHEVAL DE VOTRE CHOIX"
1240 PRINT"PARIEZ AUTANT DE FOIS QUE"
1250 PRINT"VOUS LE DESIREZ ,MAIS MISEZ"
1260 PRINT"QUELQUE CHOSE A CHAQUE FOIS"
1270 FOR D = 0 TO 5000 : NEXT D
1280 CLS
1290 RETURN
1500 REM SOUS-PROGRAMME : PARI
1510 REM IMPRESSION DES NOMS DE CHEVAUX
1520 FOR X = 1 TO 4
1530 PRINT: PRINT "     ";X;"     ";C$(X)
1540 NEXT X
1550 FOR X = 0 TO 31: PRINT "_";:NEXT X
1560 PRINT "LES COTES SONT
     DE 4 CONTRE 1"
1570 PRINT "SUR QUEL CHEVAL
     PARIEZ-VOUS ?"
1580 INPUT "(DONNEZ LE NUMERO)";C
1590 PRINT "IL VOUS RESTE ";K;"F"
1600 INPUT "COMBIEN MISEZ-VOUS";M
1610 IF M > K THEN GOTO 1600
1620 LET K = K - M
1630 CLS
1640 RETURN
2000 REM DESSIN DE LA PISTE
2010 FOR X = 0 TO 31 :PRINT@ (1,X),"-";
     :NEXT X
2020 FOR X = 0 TO 31 :PRINT(11,X),"-";
     :NEXT X
2030 FOR F = 1 TO 11 :
     PRINT@(F,31),CHR$(133);:NEXT F
2040 RETURN
2500 REM COURSE
2510 LET G = 0: REM VARIABLE DU GAIN
2520 REM POSITIONS COLONNE DE DEPART
2530 FOR X = 1 TO 4 : LET CO(X) = 0 :
```

```
     NEXT X
2540 REM BOUCLE DE LA COURSE
2550 FOR X = 1 TO 4
2560 PRINT@ ( RA(X),CO(X) ),CHR$(137);
2570 NEXT X
2580 FOR D = 0 TO 50 : NEXT D
2590 FOR X = 1 TO 4
2600 PRINT@ ( RA(X),CO(X) )," ";
2610 NEXT X
2620 FOR X = 1 TO 4
2630 LET CO(X) = CO(X) + 1 + INT(RND*3)
2640 IF CO(X) > 31 THEN LET CO(X) = 31:
     LET G = 1
2650 NEXT X
2660 IF G = 1 THEN GOSUB 2800: RETURN
2670 GOTO 2540
2800 REM DEPLACEMENT GAGNANT
2810 FOR X = 1 TO 4
2820 PRINT@ (RA(X),CO(X)),CHR$(137);
2830 NEXT X
2840 FOR D = 0 TO 500: NEXT D
2850 CLS
2860 RETURN
3000 REM SOUS-PROGRAMME : RESULTAT
3010 REM LE JOUEUR A-T-IL GAGNE
3020 LET G = 0 : REM VARIABLE DU GAIN
3030 FOR X = 1 TO 4
3040 IF CO(X) = 31 THEN IF C = X
     THEN LET G = 1
3050 NEXT X
3060 IF G = 1 THEN GOSUB 3300
3070 IF G = 0 THEN GOSUB 3400
3080 LET K = K + M
3090 RETURN
3300 REM SOUS-PROGRAMME: JOUEUR GAGNANT
3310 LET M = M*4
3320 PRINT@ (6,10),"FELICITATIONS"
3330 PRINT C$(C);" A GAGNE LA COURSE"
3340 PRINT "VOUS AVEZ GAGNE ";M;"F"
3350 FOR D = 0 TO 1000 : NEXT D
3360 CLS
3370 RETURN
3400 REM SOUS-PROGRAMME: JOUEUR PERDANT
3410 LET M = 0
3420 PRINT@ (6,10),"PAS DE CHANCE"
3430 PRINT C$(C);" N'A PAS GAGNE"
3440 FOR D = 0 TO 1000 : NEXT D
3450 CLS
3460 RETURN
```

```
3500 REM SEQUENCE FINALE
3510 IF K = 0 THEN GOTO 3600
3520 IF K < 50 THEN GOTO 3700
3530 IF K > 50 THEN GOTO 3800
3600 REM LE JOUEUR N'A PLUS D'ARGENT
3610 PRINT@ (6,10),"VOUS N'AVEZ PLUS
     D'ARGENT"
3620 PRINT "VOUS AUREZ PLUS DE CHANCE"
3630 PRINT "LA PROCHAINE FOIS"
3640 END
3700 REM LE JOUEUR A PERDU, MAIS
     IL LUI RESTE DE L'ARGENT
3710 PRINT@ (6,10),"PAS FAMEUX"
3720 PRINT "VOTRE CAPITAL EST DE";K;"F"
3730 PRINT "VOUS AUREZ PLUS DE CHANCE"
3740 PRINT "LA PROCHAINE FOIS"
3750 END
3800 REM LE JOUEUR A GAGNE DE L'ARGENT
3810 PRINT@ (6,10),"BIEN JOUE"
3820 PRINT "VOUS ARRETEZ ALORS QUE"
3830 PRINT "VOUS GAGNEZ"
3840 PRINT "VOUS POSSEDEZ MAINTENANT "
     K;"F"
3850 END
```

Contrôle

Nous n'avons pu vous donner, dans ce chapitre qu'un bref aperçu d'un sujet très vaste. Il se peut donc fort bien que vous ne vous sentiez pas encore suffisamment confiant pour écrire de longs programmes. Vous préférerez peut-être vous intéresser à des programmes écrits par d'autres (comme ceux que vous pourrez trouver au chapitre suivant), avant d'en écrire vous-même.

Arrivé à la fin de ce chapitre, vous devriez savoir :

Sur la structure des programmes

— le concept de programmation structurée et la division d'un programme en modules aisément maniables ;
— les méthodes simples d'assemblage des modules, à l'aide de sous-programmes ;
— la présentation d'un programme structuré (à l'aide de REM) ;
— la distinction entre variable de programme et variable de sous-programme.

Sur les instructions GOSUB... RETURN
— comment elles permettent l'appel d'un sous-programme et le retour au programme principal.
Sur les organigrammes
— comment réaliser un ensemble d'organigrammes pour faciliter la programmation structurée.
Sur le contrôle
— importance des résultats intermédiaires dans les programmes de calcul ;
— choix de valeurs simples et estimations approximatives pour tester les résultats fournis par un programme.
Sur l'utilisation de la mémoire
— réaction de l'ordinateur lorsque la mémoire arrive à saturation ;
— techniques simples pour réduire la taille d'un programme.

8

Pour aller plus loin

Au cours de ce chapitre, nous étudierons :
— la façon dont l'ordinateur fonctionne avec le Basic,
— les caractéristiques principales des autres langages.

Maintenant vos compétences devraient vous permettre d'écrire des programmes simples. Quelle sera la prochaine étape ? Nous allons essayer, dans ce dernier chapitre, de vous présenter les choix qui s'offrent à vous.

Toutes les possibilités du Basic n'ont certes pas été développées dans ce manuel. En effet, nous nous en sommes tenus à un noyau de base, commun à tous les ordinateurs, qui n'offre donc pas le même éventail d'instructions qu'une version Basic à part entière. C'est votre manuel d'utilisation qui, en premier lieu, vous permettra d'élargir cet éventail.

Si vous désirez faire de la programmation et si vous ne possédez pas d'ordinateur, il faut, avant toute chose, vous en procurer un ; autrement, vous ne pouvez espérer devenir un bon programmeur.

Il existe un grand nombre d'ouvrages spécialisés qui traitent de la programmation en Basic sur la plupart des ordinateurs familiaux. Nous pensons que la meilleure façon de vous perfectionner consiste à vous procurer un ouvrage traitant spécialement de votre micro-ordinateur. Par souci de simplification, nous avons été amenés à passer sous silence de nombreuses commandes, ainsi que des possibilités — son, couleur, par exemple — qui font tout l'attrait des micro-ordinateurs. Si vous possédez une machine récente, ou de faible diffusion, vous éprouverez sans doute quelques difficultés à trouver un livre qui lui corresponde. Nous vous suggérons la lecture de quelques ouvrages généraux qui traitent du Microsoft Basic, à la page 184. Si votre

machine ne fonctionne pas sur Microsoft, alors un ouvrage traitant d'un appareil différent du vôtre mais utilisant un Basic semblable (PET Basic ou ATARI Basic, par exemple) serait préférable.

On peut apprendre énormément en lisant ou en essayant les programmes des autres, même si l'on n'approuve pas complètement leur façon de programmer. Un grand nombre de magazines consacrés aux ordinateurs reproduisent des listages de programmes. Certains sont spécialisés et ne traitent que d'ordinateurs particuliers. Parcourez-les et voyez à quelles machines les exemples de programmes sont destinés. Si votre ordinateur ne s'apparente pas au modèle cité, attendez-vous à devoir effectuer de nombreuses modifications.

Repérez les endroits où vous pourriez rencontrer des difficultés : le programme, par exemple, peut être destiné à un écran différent du vôtre, il utilise une commande dont votre Basic est démuni... Avant de vous fatiguer à entrer le programme, assurez-vous bien que vous saurez surmonter les difficultés rencontrées.

L'achat de programmes tout faits, sur cassette ou sur disquette, vous aidera à évaluer les capacités de votre ordinateur. Ne pêchez pas par excès d'orgueil en essayant de tout faire vous-même — investissez dans quelques bons programmes de professionnels. Mais n'espérez pas pouvoir profiter des « trucs » qu'ils comportent. La plupart des programmeurs ont recours à des astuces qui empêchent de lister et de « pirater » leurs programmes.

Tout en progressant dans votre étude du Basic, vous en découvrirez les atouts et les limites. En voici quelques exemples :

— Le Basic est lent. Il convient tout à fait aux jeux simples et aux programmes graphiques semblables à ceux que nous avons écrits jusqu'ici. Mais sa lenteur constitue un véritable handicap pour les programmes comportant un grand nombre de calculs mathématiques, ou bien pour les programmes d'animation.

— C'est un langage qui dispose d'outils de structure relativement pauvres. En conséquence, les programmes longs se révèlent souvent difficiles à écrire, puis une fois écrits, à comprendre.

— Les possibilités de manipulation de fichiers sont très limitées. On ne peut mémoriser, sur disque ou cassette, que des fichiers de données rudimentaires.

— Le Basic ne peut être utilisé pour résoudre les

problèmes de maintenance programmée comme, par exemple, la régulation thermique d'une serre à l'aide d'un micro-ordinateur.

Lorsque l'on veut réaliser quelque chose et que le Basic s'avère inapproprié, il est nécessaire de recourir à un autre langage.

Du Basic au langage machine

Les compétences fondamentales de votre ordinateur sont très limitées. Il ne peut exécuter qu'un petit nombre d'opérations, consistant à manipuler des symboles : addition, soustraction, comparaisons. Les circuits qui permettent à l'ordinateur d'effectuer ces opérations sont situés dans l'unité centrale (la « puce » pour les micro-ordinateurs).

Les constructeurs d'ordinateurs élaborent, à partir de cet ensemble d'opérations de base, des programmes permettant à l'ordinateur d'accomplir d'autres tâches, comme, par exemple, le décodage des touches du clavier et l'affichage des informations sur l'écran. Les instructions qui permettent d'effectuer ce genre d'opérations constituent le « système d'exploitation ». Si votre ordinateur peut utiliser les données stockées sur une cassette et s'il peut à son tour stocker des données sur celle-ci, alors il dispose d'un « système d'exploitation à cassette » ; s'il fonctionne sur disquettes ou disques durs, alors, il possède un « système d'exploitation à disque ». Habituellement, les instructions du système d'exploitation sont intégrées à l'ordinateur et se trouvent dans sa mémoire morte (MEM). Quelquefois, les systèmes d'exploitation à disque doivent être, au moins partiellement, chargés à partir d'une disquette.

Actuellement, les systèmes d'exploitation les plus utilisés sur micro-ordinateurs sont : CP/M, MS/DOS et UNIX.

Peut-être avez-vous déjà utilisé, sans vous en rendre compte, des commandes dépendant directement du système d'exploitation. C'est le cas pour LOAD et SAVE ; elles n'appartiennent pas, à proprement parler, au langage Basic.

Le système d'exploitation constitue une bonne partie de ce qu'il est convenu d'appeler le logiciel de l'ordinateur. Ce logiciel comporte également un interpréteur, puisque vous possédez un ordinateur qui comprend le Basic. Ce programme spécial peut tout à fait s'identifier à un traducteur humain. En effet, le Basic, comme tous les langages évolués, n'est pas un langage immédiatement compréhensible par l'ordina-

teur. L'interpréteur assure la traduction du Basic en langage machine (constitué des assemblages de 0 et de 1 dont nous avons parlé dans l'introduction). Ainsi, chaque instruction Basic est transformée en une série d'instructions produites par l'unité centrale, en adresses mémoire et en données avec lesquelles l'ordinateur peut travailler directement.

Programmer en langage d'assemblage

Bien que l'interpréteur fasse du très bon travail, il est bien plus rapide et efficace de s'adresser directement à l'ordinateur dans son propre langage. Le seul inconvénient, c'est que l'apprentissage du langage de l'ordinateur est nettement plus ardu. Néanmoins, si vous programmez beaucoup, il vous sera sans doute nécessaire de travailler en code machine, peut-être en utilisant un programme d' « assemblage » qui autorise les abréviations, ou mnémoniques, comme ADD ou LDA (signifiant respectivement « ajouter le nombre dans le totalisateur à celui que je vais donner dans l'accumulateur » ou « mémoriser ce nombre dans l'accumulateur »). Le passage au langage machine s'effectue ensuite rapidement grâce à l' « assembleur », programme résidant en mémoire morte.

Autres langages « évolués »

Même lorsque l'on devient très compétent, la programmation en code machine est fastidieuse.
Une solution permettant d'éviter les imperfections du Basic consiste à utiliser des langages de haut niveau dits « évolués ». Comme le Basic, ce sont des langages conçus pour donner des instructions à l'ordinateur par l'intermédiaire de leurs propres interpréteurs, ou de programmes similaires, appelés « compilateurs » (voir ci-dessous). Voici quelques-uns des langages les plus courants acceptés par les ordinateurs domestiques :
Le *Forth,* nouveau langage très à la mode. Malgré la lourdeur de la notation polonaise, utilisée pour les fonctions arithmétiques, ce langage possède un grand avantage : la rapidité.
Le *Pascal,* langage structuré, convient parfaitement à la rédaction de programmes longs.
Le *Logo,* langage simple (du moins dans certaines versions), particulièrement utile pour aider les enfants à se familiariser avec l'informatique.
Si vous avez l'intention d'écrire des programmes bien spécifiques, vous aurez intérêt à rechercher un langage adapté à ce que vous envisagez de réaliser. Parlez-en à votre revendeur ou consultez des magazines informatiques.

Utiliser un compilateur

Pour finir, quelques remarques sur les compilateurs. Programmes spéciaux de traduction, ils se distinguent des interpréteurs par la manière dont ils transcrivent les instructions en langage machine. L'interpréteur traduit les lignes de programme (et contrôle qu'elles ne contiennent pas d'erreurs) une à une, et les exécute également une à une. Le compilateur traduit et exécute le programme en une fois, de la première à la dernière ligne.

Le Basic est un langage interprété, il est beaucoup plus lent qu'un langage compilé, mais il est plus facile à mettre au point ; lorsqu'il ne fonctionne pas, ou lorsqu'il nécessite des améliorations, l'édition en est extrêmement simple. En revanche, si un langage compilé permet d'accélérer l'exécution d'un programme, son édition n'est pas aussi facile. Il en résulte que les langages qui sont avant tout des langages compilés comme le FORTRAN, sont d'un maniement plus délicat.

Cependant, il est possible de se procurer des compilateurs fonctionnant sur des programmes en Basic. Pour les utiliser, il suffit d'abord de mettre au point le programme, grâce à l'interpréteur, et ensuite de le faire passer par le compilateur. On exécute alors la version compilée, en conservant le programme source (c'est-à-dire ce que l'on a écrit originellement), pour pouvoir le consulter ou le modifier ultérieurement. Le compilateur Basic répond ainsi aux exigences de rapidité.

La conclusion de tout ceci c'est qu'il vous reste encore du chemin à faire ! En fait vous n'en êtes qu'au tout début du voyage, et la partie la plus intéressante est à venir, car, maintenant vous êtes en mesure de concevoir et d'écrire vos propres programmes.

Quelques programmes de plus

Nous n'allons pas nous livrer avec vous à l'étude détaillée de nouveaux programmes. Nous nous contenterons de vous fournir les listages de quelques programmes assez courts. Ils n'utilisent que les instructions Basic que nous avons étudiées ensemble. Exploitez-les, adaptez-les, enrichissez-les d'idées personnelles, ou de trucs puisés dans la lecture de listages. Vous écrirez alors des programmes qui vous serviront ou qui vous amuseront pendant des années.

Histogramme

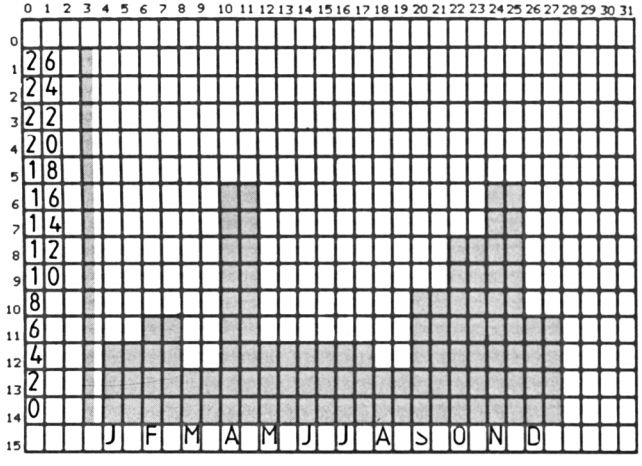
Ce programme réalise un diagramme à barres simple en utilisant des données mensuelles. Il met automatique-

ment les données à l'échelle appropriée (de façon assez grossière), en prenant comme référence la plus haute valeur. Vous verrez qu'il fonctionne mieux pour les valeurs entières inférieures à 100.

Variables utilisées

Tableau des données mensuelles	M (1 à 12)
Valeur la plus grande	G
Boucles de comptage	X,N
Échelle	E
Axe horizontal	H
Axe vertical	V
Ordonnées (graduations verticales)	O
Colonne représentant chaque donnée	C

Exemple d'histogramme




```
10  REM *** HISTOGRAMME ***
20        PAR SUSAN CURRAN
30  DIM M(12) : REM DONNEES MENSUELLES
40  PRINT "ENTRER LES DONNEES MOIS PAR
    MOIS"
50  PRINT "EN COMMENCANT PAR JANVIER"
60  FOR X = 1 TO 12
70  INPUT M(X)
80  NEXT X
90  REM TROUVER LA DONNEE LA PLUS
    GRANDE
100 LET G = 0
110 FOR X = 1 TO 12
120 IF M(X) > G THEN LET G = M(X)
130 NEXT X
```

```
140 LET E = INT(G/13)+1 : REM ECHELLE
150 REM TRACE DES AXES
155 CLS
160 FOR H = 5 TO 28
170 PRINT @ (13,H), CHR$(128); : REM
    AXE HORIZONTAL
180 NEXT H
190 PRINT @ (14,5), "J F M A M J J A S
    O N D"
200 LET O = E*13 : REM GRADUATION
210 FOR V = 0 TO 13
220 PRINT @ (V,4), CHR$(133); : REM
    AXE VERTICAL
230 PRINT @ (V,0),O;
240 LET O = O - E
250 NEXT V
300 REM TRACE DES DONNEES
310 LET C = 5 : REM COLONNES
320 FOR N = 1 TO 12
330 FOR X = 12 TO 13 - (INT(M(N)/E))
    STEP -1
340 PRINT @ (X,C), CHR$(128)+CHR$(128);
350 NEXT X
360 LET C = C + 2
370 NEXT N
380 GOTO 380
```

Solde de compte-chèque

Un programme quelque peu rudimentaire, que vous ne devriez pas avoir de difficultés à améliorer.

Variables utilisées

Solde	S
Débit	D
Crédit	C

```
10  REM *** SOLDE DE COMPTE-CHEQUE ***
20  INPUT "ENTRER LE DERNIER SOLDE";S
30  PRINT "ENTRER LES MONTANTS RETIRES"
40  PRINT "APRES LE DERNIER MONTANT"
50  PRINT "ENTRER 0"
60  INPUT D
70  LET S = S - D
80  IF D = 0 THEN GOTO 100
90  GOTO 50
100 PRINT "ENTRER LES MONTANTS
    CREDITES"
110 PRINT "APRES LE DERNIER MONTANT"
120 PRINT "ENTRER 0"
130 INPUT C
140 LET S = S + C
```

```
150 IF C = 0 THEN GOTO 170
160 GOTO 130
170 PRINT "POSITION ACTUELLE : ";S
```

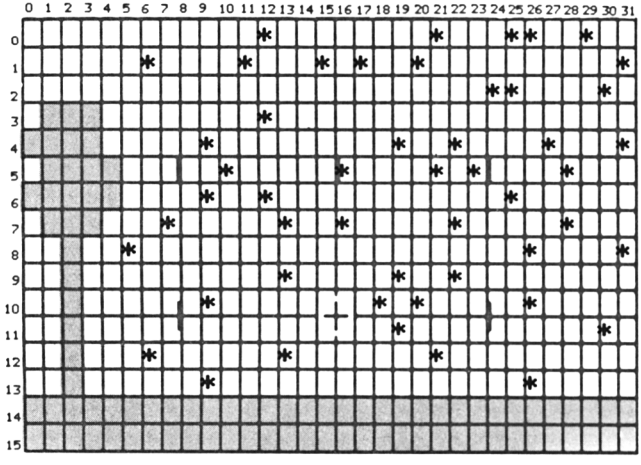
Clair d'étoiles

Programme graphique très simple à réaliser.

Variables utilisées

Numéro de rangée	R
Numéro de colonne	C
Chaînes de caractères graphiques	
X2	A$
X3	B$
Compteur de boucles	N

Clair d'étoiles




```
10  REM *** CLAIR D'ETOILES ***
20  CLS
30  LET A$ = CHR$(128)+CHR$(128)
40  LET B$ = CHR$(128)+CHR$(128)+
    CHR$(128)
50  REM DESSIN DU SOL
60  FOR R = 14 TO 15
70  FOR C =  0 TO 31
80  PRINT @ (R,C), CHR$(128);
90  NEXT C
100 NEXT R
110 REM DESSIN DE L'ARBRE
120 FOR R = 8 TO 13
```

```
130 PRINT @ (R,2), CHR$(128);
140 NEXT R
150 PRINT @ (3,1), B$;
160 PRINT @ (4,0), A$+A$;
170 PRINT @ (5,1), A$+A$
180 PRINT @ (6,0), A$+B$;
190 PRINT @ (7,1), B$;
200 REM APPARITION DES ETOILES
210 FOR N = 0 TO 50
220 PRINT @ (INT(RND*14),INT(RND*27)+5),
    "*";
230 NEXT N
240 GOTO 240
```

La fusée

Notez comment la flamme de la fusée s'efface sous une traînée d'étincelles...

Variables utilisées

Numéros de rangée	R
Numéros de colonne	C
Chaînes de caractères graphiques	
X1	A$
X2	B$

```
10  REM *** FUSEE ***
20  LET A$ = CHR$(128)
30  LET B$ = CHR$(128)+CHR$(128)
40  LET R = 12 : LET C = 1
50  CLS
60  FOR X = 10 TO 0 STEP -1
70  PRINT @ (5,15), X;
80  FOR D = 0 TO 460 : NEXT D
90  NEXT X
100 PRINT @ (5,10), "FEU!!"
110 PRINT @ (R,C+1), B$;
120 PRINT @ (R+1,C), A$+B$;
130 PRINT @ (R+2,C-1), "  ";A$;
140 PRINT @ (R+3,C)," ";
150 LET R = R-1 : LET C = C+1
160 IF R = -1 THEN GOTO 180
170 GOTO 110
180 CLS : END
```

Agenda

Un exemple d'utilisation des tableaux : la création d'un agenda. Puisque les données sont mémorisées dans le programme sous forme d'instructions DATA, elles ne sont pas supprimées quand on réexécute le pro-

gramme. Pour changer les données, retapez simplement les instructions DATA. Notez la nécessité de « remplir » ces instructions en complétant par des zéros. Nous n'avons pas employé auparavant l'expression OR (ou), mais la façon dont elle est utilisée ici en éclaire pleinement le sens. Exercez-vous à terminer ce programme.

Variables

Tableaux de données	
Dates (jour, mois, année)	D1$ (6)
Événements, rendez-vous, etc.	O1$ (6)
Dates (jour, mois seulement)	D2$ (6)
Événements, rendez-vous, etc.	O2$ (6)
Dates (jour seulement)	D3$ (6)
Événements, rendez-vous, etc.	O3$ (6)
Date choisie par l'usager	
Jour	C1$
Mois	C2$
Année	C3$
Compteurs de boucles	X

```
10  REM *** AGENDA ***
20  REM PAR SUSAN CURRAN
30  CLS
40  REM TABLEAUX DE DONNEES
50  DIM D1$(6) : DIM O1$(6)
60  FOR X = 1 TO 6
70  READ D1$(X) : READ O1$(X)
80  NEXT X
90  DIM D2$(6) : DIM O2$(6)
100 FOR X = 1 TO 6
110 READ D2$(X): READ O2$(X)
120 NEXT X
130 DIM D3$(6): DIM O3$(6)
140 FOR X = 1 TO 6
150 READ D3$(X): READ O3$(X)
160 NEXT X
170 REM OBTENIR LA DATE DESIREE
180 INPUT "ENTRER LA DATE SOUS LA FORME
    JJ,MM,AA";C1$,C2$,C3$
190 PRINT "VOTRE PROGRAMME
    AUJOURD'HUI : "
200 REM PASSAGE EN REVUE TABLEAU D1
210 FOR X = 1 TO 6
220 IF C1$+C2$+C3$ = D1$(X) THEN PRINT
    O1$(X)
230 NEXT X
240 REM PASSAGE EN REVUE TABLEAU D2
```

```
260 IF C1$+C2$ = D2$(X) THEN PRINT
    O2$(X)
270 NEXT X
280 REM TABLEAU D3 + VERIF. FIN DE MOIS
290 FOR X = 1 TO 6
300 IF C1$ = D3$(X) THEN PRINT O3$(X) :
    GOTO 340
310 IF C1$ ="28" THEN IF C2$ ="02" THEN
    IF D3$(X) ="FIN" THEN PRINT O3$(X):
    GOTO 340
320 IF C1$ ="30" THEN  IF C2$ ="04" OR
    IF C2$ ="06" OR IF C2$ ="09" OR
    IF C2$ ="11" THEN IF D$(X) ="FIN"
    THEN PRINT O3$(X): GOTO 340
330 IF C1$ ="31" THEN IF D3$(X) ="FIN"
    THEN PRINT O3$(X) :GOTO 340
340 NEXT X
350 PRINT "C'EST TOUT" :END
400 REM DONNEES POUR UNE ANNEE PRECISE
410 DATA "020284","SUSAN DENTISTE 10.30",
    "030284","CLUB INFORMATIQUE 7.30",
    "0440584","ACHEVER CHAPITRE 7"
420 DATA "110285","SIGNATURE CONTRAT
    7.30","0","0","0","0"
430 REM DONNEES POUR L'ANNEE EN COURS
440 DATA "2603","ANNIVERSAIRE DE JEAN",
    "1405","ANNIVERSAIRE DE SUSAN"
450 DATA "0711","ANNIVERSAIRE DE RUFUS",
    "1412","ANNIVERSAIRE D'EVA"
460 DATA "3112","RENOUVELLEMENT DE LA
    REDEVANCE TV","33006","VERIFIER
    CHAUFFAGE CENTRAL"
470 REM DONNEES POUR LE MOIS EN COURS
```

Annexe 1 Des ordinateurs et des Basics

Mise en page des programmes

Commençons par quelques remarques sur la façon dont les programmes sont présentés dans cet ouvrage. En général, ils sont « aérés » ; les mots sont séparés par des espaces, que vous n'êtes pas obligé de reproduire. Dans les premier et deuxième chapitres, nous avons délibérément inséré des espaces, pour des raisons de présentation. Nous n'en avons pas précisé le nombre, mais, puisqu'un blanc occupe exactement la même place qu'un caractère (chiffre ou lettre), il n'est pas difficile d'en faire le compte.

La présentation des lignes de programme à l'écran se distingue à deux égards de leur mise en page dans l'ouvrage. Le chiffre zéro représenté par un 0, est rendu sur l'écran par un 0 barré : Ø. Ensuite, dans un but de clarté et de lisibilité, nous avons bien mis en évidence les numéros de ligne sur la gauche et nous avons fractionné les lignes de programme qui nous paraissaient trop longues. Sur votre écran, il vous semblera que les lignes se ramassent sur le côté gauche.

Les versions Basic

D'un micro-ordinateur à l'autre, nombreuses sont les différences : matériel, clavier, système d'exploitation, version Basic, commandes relatives aux fonctions non couvertes par le Basic...

Nous avons voulu écrire un livre avant tout orienté vers les applications pratiques, c'est pourquoi nous avons utilisé six ordinateurs différents. Cette annexe servira deux objectifs. D'abord, nous traiterons des caractéristiques des différentes versions Basic, dans l'ordre de leur apparition dans les chapitres. Ensuite, nous passerons rapidement en revue les ordinateurs que nous avons utilisés, ainsi que les spécificités de leur Basic.

Nous n'insisterons jamais assez sur le fait que cet ouvrage ne peut se substituer au manuel d'utilisation de votre ordinateur ; c'est lui qui vous fournira des explications détaillées sur le fonctionnement de votre machine. Nous nous contentons de vous indiquer où y trouver les renseignements dont vous avez besoin.

Chapitre 1

Majuscules et minuscules Tous les micro-ordinateurs que nous avons essayés acceptent les instructions Basic en majuscules. Dans certains cas, elles sont obtenues par la pression d'une seule touche. Le problème est alors résolu ! Seul un petit nombre d'ordinateurs accepte les minuscules pour les instructions, ainsi que, dans la plupart des cas, les noms de variables, les chaînes, etc.

Manipulation des nombres Si vous avez l'intention d'effectuer de nombreux calculs, alors la précision de votre ordinateur revêtira la plus grande importance. Cette précision varie suivant la taille des nombres qu'il peut accepter. Par exemple 0,0000234552334 est un très petit nombre traité avec une précision comparativement grande. Il y a neuf chiffres significatifs. Une précision de six chiffres peut être considérée comme médiocre ; en revanche, une précision de neuf, dix chiffres est tout à fait appréciable. Tous les ordinateurs adoptent la notation scientifique pour afficher petits et grands nombres. C'est l'espace dont ils disposent pour les stocker qui détermine le recours à cette notation et leur précision.

Les ordinateurs n'utilisent pas tous le signe + implicite. Dans certains cas, les nombres séparés par des points-virgules sont juxtaposés à l'affichage.

Les fonctions arithmétiques Certains ordinateurs en proposent beaucoup, d'autres peu. Nous les utilisons rarement dans cet ouvrage.

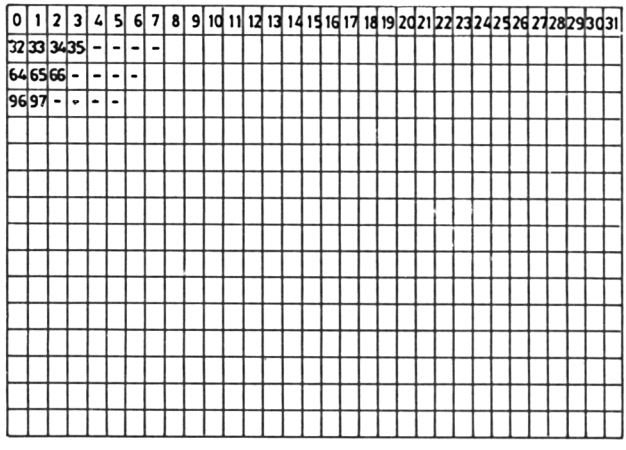
CLS Les ordinateurs possèdent tous une commande qui efface l'écran. Ce n'est pas toujours CLS. Consultez votre manuel ainsi que nos paragraphes relatifs aux instructions PRINT.

Syntaxe générale Notre version Basic est acceptée par la quasi-totalité des ordinateurs, mais consultez votre manuel — à la rubrique PRINT, éventuellement — si vous avez des problèmes.

PRINT @ C'est sans doute le mot clé le plus diversement traitée par les versions Basic. Voici quelques remarques générales qui vous guideront :

— Les exigences de la syntaxe (parenthèses, virgules, points-virgules, etc.) varient considérablement. Consultez l'index de votre manuel à PRINT @. Sur certains ordinateurs, le symbole @ est remplacé par AT.
— Beaucoup d'ordinateurs n'utilisent qu'un seul nombre pour décrire la position d'impression. L'écran est alors numéroté de gauche à droite, rangée par rangée, comme sur la figure suivante.

Autre forme de codage de la grille d'écran
Certains écrans sont numérotés successivement, rangée par rangée, comme ceci :



Il n'est pas difficile de transformer les numéros de rangée et de colonne que nous utilisons en un seul nombre ; voici comment procéder :
Prenez le numéro de rangée et multipliez-le par le nombre d'emplacements contenus dans une ligne d'écran. Par exemple, pour imprimer sur la rangée 11 d'un écran de 32 colonnes, effectuez le calcul suivant :

```
11*32
```

Ensuite, ajoutez le nombre représentant le numéro de colonne. Ainsi pour imprimer à la colonne 10 de la rangée 11, vous obtiendrez :

```
11*32+10
```

Vous pouvez insérer l'expression telle quelle dans une instruction PRINT @ et laisser l'ordinateur faire le calcul lui-même :

```
PRINT @ (11*32+10), "RANGÉE 11, COLONNE 10"
```

Lorsque l'on utilise des variables dans les instructions PRINT @, il semble plus pratique d'avoir recours, comme nous, à deux variables.
Sur certains ordinateurs, cette instruction n'existe pas. Il est certainement possible alors de recourir à une autre méthode — sans doute plus fastidieuse.

Même si vous ne possédez pas l'une des machines que nous décrirons, le fait de savoir comment elles manipulent les positions PRINT @ vous permettra peut-être de comprendre la façon dont votre ordinateur procède. Sur beaucoup de machines similaires à celles que nous présentons, au lieu de recourir à PRINT @, on utilise une série d'instructions qui placent le curseur à l'endroit désiré.
Certains ordinateurs disposent de systèmes d'exploitation de l'écran assez complexes, mais les commandes qui, à première vue, peuvent paraître obscures offrent de nombreuses possibilités comme, par exemple, l'effacement de l'écran ou la remise du curseur à sa position initiale dans le coin supérieur gauche.

Chapitre 2

END Pratiquement toutes les versions Basic en disposent. END est parfois remplacée par STOP (dont le rôle est très légèrement différent).
LIST La syntaxe utilisée pour le listage des lignes est variable. Consultez votre manuel si votre ordinateur n'accepte pas celle que nous employons.
NEW Sur certains ordinateurs, cette commande effacera aussi bien la mémoire principale que l'écran.
SAVE et *LOAD* Ces commandes appartiennent à proprement parler au système d'exploitation de l'ordinateur, pas à son Basic. Il existe un grand nombre de variantes : CSAVE, CLOAD, CHAIN, entre autres. On trouve également des commandes très utiles comme SKIFF et VERIFY. Voyez dans votre manuel la partie qui traite de l'utilisation du lecteur de cassettes.
BREAK Les ordinateurs ne disposent pas tous de la touche BREAK. Mais il existe toujours une autre manière d'interrompre le programme. Voyez la rubrique « interruption » (« interrupt » en anglais) dans votre manuel.
Les caractères graphiques La plupart (mais pas la totalité) des ordinateurs domestiques disposent de caractères graphiques identiques ou semblables aux nôtres. Si le vôtre n'en possède pas, servez-vous des touches caractères (lettres, nombres, etc.). Plus tard,

vous apprendrez à créer vous-même vos propres caractères. Sur certains micro-ordinateurs, il est possible de choisir les caractères graphiques en appuyant simplement sur une touche prévue à cet effet, puis sur une touche du clavier AZERTY (ou QWERTY) ; il n'est alors plus obligatoire de recourir aux codes CHR$.

Chapitre 3

LET Dans la plupart des cas son utilisation n'est pas impérative. « LET C = 10 » équivaut à « C = 10 ».
Les noms de variable Beaucoup d'ordinateurs offrent la possibilité d'utiliser des noms plus longs et plus « parlants », comme « ADRESSE » ou « TOTAL », ce qui facilite la lecture des programmes. Quelques ordinateurs ne prennent en compte que les deux premiers caractères ; aussi « TOTAL » et « TOUCHE » seraient interprétées comme étant la même variable. Beaucoup emploient différents types de variables numériques, selon qu'il s'agit de nombres entiers, de nombres décimaux... Les noms simples que nous utilisons dans l'ouvrage sont couramment acceptés.
INPUT La syntaxe varie, plus particulièrement lorsque vous entrez deux articles ou plus à la fois. Consultez votre manuel.
Les nombres aléatoires Une fois encore, la syntaxe est variable. Certains ordinateurs possèdent une instruction qui leur permet de générer des nombres entiers aléatoires. Étant donné que l'emploi en est plus simple, vous ne devriez pas rencontrer de difficultés. Il se peut que vous soyez amené à employer l'instruction RANDOMIZE qui influe sur l'ordre dans lequel les nombres sont générés.

Chapitre 4

La vitesse Notez que tous les ordinateurs ne travaillent pas à la même vitesse ! Une boucle de temporisation de 0 à 460 demande une seconde de délai, par exemple, sur l'ordinateur TANDY couleur. Il vous faudra peut-être choisir des nombres différents sur votre ordinateur.

Chapitre 5

Beaucoup d'ordinateurs offrent des commandes spéciales pour l'édition et le débogage. Nous ne les avons pas utilisées dans ce livre.

Chapitre 6

Les tableaux Nous n'avons eu recours qu'à des tableaux unidimensionnels. Un grand nombre d'ordinateurs permettent d'utiliser deux dimensions et plus, ce qui est très commode. Si vous utilisez de grands tableaux, vous devrez leur réserver de l'espace mémoire. La commande CLEAR permet souvent de le faire. Notez que les ordinateurs SINCLAIR manient les tableaux de manière légèrement différente.

READ... DATA Ces deux instructions sont quasiment universelles, mais elles faisaient défaut aux premiers SINCLAIR, absence très critiquée. Si vous possédez l'un de ces modèles, utilisez LET et modifiez votre programme en conséquence.

Divers

Nous avons laissé de côté la couleur et le son, non pas parce qu'ils sont d'un maniement difficile, mais parce que le mode d'utilisation en est trop variable. Si votre machine vous offre les possibilités du son et de la couleur, étudiez leur fonctionnement. Au prix d'un petit effort vous pourrez ainsi agrémenter un grand nombre de vos programmes.

Bien entendu, il existe encore d'autres instructions Basic dont nous n'avons pas parlé. Mais, encore une fois, nombre d'entre elles varient d'une machine à l'autre. Bon nombre d'ordinateurs permettent de raccourcir les programmes : omissions de mots, abréviations des commandes, etc. Pour plus de précisions, consultez votre manuel.

Les machines que nous avons utilisées

Nous avons essayé toute une gamme de machines de diverses marques. Celles que nous avons retenues sont parfaitement représentatives des différences existant entre les ordinateurs domestiques les plus courants.

Le Commodore 64 Ce micro-ordinateur est doté de possibilités graphiques et sonores étonnantes, mais la maîtrise de ces caractéristiques séduisantes est malaisée, car le Basic résident est assez pauvre. On pourra néanmoins se procurer un logiciel spécifique, le « TOOL », comprenant les instructions manquantes, et qui se révélera un instrument très utile. Sur le Commodore 64, il est possible d'inclure des commandes — que l'on obtient, en mode direct, en appuyant sur une touche spéciale — dans des chaînes. Pour effacer l'écran au cours d'un programme, il suffit de placer la

commande « CLEAR » dans une chaîne, à l'aide d'une instruction de ce type :

```
PRINT "(touche CLEAR)"
```

Il n'y a pas d'instruction PRINT @. On se sert de commandes de curseur dans des chaînes, combinées avec TAB. Des exemples sont fournis dans le manuel d'utilisation, mais cette instruction est d'un maniement délicat et vous préférerez peut-être adapter nos programmes et utiliser plus souvent la seule instruction PRINT. La longueur de ligne du Commodore 64 est assez réduite. Au-delà de 80 caractères, il produira un message d'erreur. Un éloge, pour finir : il dispose d'un très bon éditeur d'écran.

Le Dragon 64 Il fonctionne sur la base d'une version Basic Microsoft similaire à la nôtre. Il vous faudra convertir nos numéros de rangée/colonne en un seul nombre et modifier les commandes R et C. Il est nécessaire de réserver de l'espace mémoire pour tous les tableaux — sauf les petits — à l'aide de CLEAR. Les instructions couleur et son se manipulent aisément. Par ailleurs, la numérotation des lignes est facultative.

Alice et Alice 90 Leur Basic est une version simplifiée de Microsoft. S'il lui manque certaines instructions graphiques, son clavier offre, en revanche, la possibilité d'entrer directement les mots-clés du Basic ainsi que certains caractères semi-graphiques. Vous devrez convertir les numéros de rangée/colonne en un seul chiffre après PRINT @. La fonction RND est suivie de parenthèses : RND (6) et non RND*6. Consultez le manuel, extrêmement bien présenté pour y découvrir les autres instructions. Dans cet ouvrage, les programmes listés sur imprimante ont été entrés sur Alice.

Le ZX Spectrum de Sinclair Il est plus puissant et plus souple que le ZX81, mais il utilise le même Basic. Chaque instruction est entrée d'un bloc, par la pression d'une seule touche. Si l'on tape le mot clé lettre par lettre, on n'obtient rien.
Le ZX Spectrum manie les tableaux de chaîne de manière inhabituelle : tous les articles doivent être d'une longueur bien déterminée. Cela pose quelques difficultés dans notre programme « Bibliothèque » de la page 123, mais il est possible de les surmonter grâce à une bonne gestion des espaces. Son utilisation de l'écran est également inhabituelle, puisqu'elle vous

réserve les lignes du bas lorsque vous entrez du texte. Il est possible de réaliser de bonnes images en mode direct.
Tout bien pesé, c'est au Basic utilisé par le Spectrum que va notre préférence et nous vous le recommandons. La machine se révèle très efficace dans la détection et le signalement des erreurs. Les programmes sont donc faciles à lister et à éditer. Le Basic présente également un jeu complet d'instructions sonores et graphiques qui facilitent le tracé de courbes, droites, cercles, etc.

Le Tandy TRS Color / Color 2 Presque un Dragon, mais avec un nom différent. De moindre puissance, son Basic est plus limité. La version Basic étendu — que nous n'utiliserons pas ici — est identique au Basic du Dragon. C'est une machine facile à utiliser par les débutants, et sur laquelle tous nos programmes fonctionnent avec un minimum de modifications. Cependant, les instructions graphiques sont d'une utilisation difficile ; on peut noter l'absence de touche de contrôle, ou ESCAPE.

L'Électron (Acorn) Le Basic de l'Électron est identique à celui du BBC : même qualité et même ingéniosité. De nombreuses instructions vous seront très utiles : ELSE, REPEAT UNTIL... L'instruction PRINT TAB remplace PRINT @. Il ne faut pas laisser d'espace après TAB et vous devrez inverser les numéros de rangée et de colonne que nous vous donnons.

Les nombreux modes utilisés par la machine BBC peuvent être déroutants pour un débutant. Exécutez nos programmes en conservant le mode 7 ; c'est celui que sélectionne l'ordinateur lorsque vous le mettez en marche. Il est difficile d'utiliser les symboles graphiques vidéotex avec PRINT TAB ; aussi, dans le cas où vous rencontreriez quelques difficultés, essayez de vous débrouiller avec le CHR$(255), carré blanc qui ne nécessite pas de codes de contrôle. N'abordez l'utilisation de la couleur que lorsque vous maîtriserez bien la machine.

Annexe 2 Version Basic utilisée dans le livre

Commentaires généraux et syntaxe

D'ordinaire, toutes les instructions Basic s'écrivent in extenso, sans abréviations, ni omissions de mots clés. Les lignes de programme sont précédées de numéros entiers compris entre 1 et 9999. Il est autorisé d'écrire plus d'une instruction par ligne ; les instructions sont alors séparées par deux points (:).

On utilise deux catégories de noms de variable :

variables numériques :	lettre capitale / lettre capitale ou nombre ; par exemple : A1
variables de chaîne :	comme plus haut, mais suivis du signe $, par exemple : BB$

Dans les tableaux, les noms sont identiques à ceux des variables, mais on leur affixe des nombres : ex. : BB$(5), A(5).

Il existe seize caractères graphiques. On les obtient à l'aide de l'expression CHR$. Les caractères et les codes sont listés à la page 51. Les espaces hors chaînes ne sont, en général, pas indispensables.

Mots clés

CLS	Efface l'écran
DATA	Introduit une liste de données Les articles sont séparés par des virgules, les chaînes sont placées entre guillemets, par exemple : DATA 1,2,3 DATA "MARIE","JEANNE","ANNE"
DIM	Dimensionne les tableaux de données, ex. DIM A(4) introduit quatre articles dans le tableau A. Les tableaux sont supposés être de base 1 (pas d'article 0). Seuls les tableaux à une dimension sont utilisés dans cet ouvrage. L'emploi de DIM est indispensable avant l'utilisation de tout tableau.

END	Provoque l'arrêt de l'exécution d'un programme. Son emploi est impératif.
FOR... TO... NEXT	Ensemble de mots-clés utilisés pour les boucles de comptage ; ex. : FOR X = 1 TO 4... NEXT X. Voir également STEP.
GOSUB	Appelle un sous-programme débutant à un numéro de ligne donné ; ex. : GOSUB 1000.
GOTO	Renvoie à une ligne donnée ; ex. : GOTO 50.
IF... THEN	Si la proposition qui suit IF est vérifiée, alors, la ou les instructions suivant THEN (et ce, jusqu'au bout de la ligne) sont exécutées. Dans le cas contraire, le programme passe directement à la ligne suivante.
INPUT	Le programme attend qu'un ou plusieurs articles soient entrés ; ils sont associés à des variables. Le mot clé peut être suivi d'un message placé entre guillemets ; ex. : INPUT "DONNEZ-MOI DEUX NOMBRES" ; A,B. L'instruction permet d'imprimer "DONNEZ-MOI DEUX NOMBRES" sur l'écran et d'associer les deux nombres aux variables A et B.
LET	Associe une valeur à une variable. L'instruction utilisée est LET (variable) = (valeur), ex. : LET A=50. Toute variable doit se voir affectée une valeur avant d'être utilisée (même si cette valeur est 0) ; on ne peut omettre LET.
LIST	Liste tout ou partie du programme. Différentes possibilités : LIST : le programme est listé dans sa totalité. LIST (ligne) : une seule ligne est listée (ex. : LIST 10). LIST (ligne)-(ligne) : liste une série de lignes (ex. : LIST 10-50). LIST (ligne)- : liste à partir d'une ligne bien précise jusqu'à la fin du programme (ex. : LIST 50-). LIST -(ligne) : liste du début du programme jusqu'à la ligne donnée (ex. : LIST -50).

LOAD	Charge un programme à partir d'une cassette. Pas étudié en détail.
NEW	Efface le programme et les variables contenus dans la mémoire vive.
NEXT	Utilisée avec FOR... TO (voir ci-dessus).
PRINT	Imprime sur l'écran les données dont on la fait suivre (nombres, chaînes ou variables). Ainsi, PRINT A imprime le contenu de la variable A et PRINT "HELLO" imprime la chaîne « HELLO ». Les virgules permettent de passer à la zone d'impression suivante (sachant que l'écran est divisé en quatre zones de huit colonnes). Les point-virgules séparent les articles à imprimer. Placés à la fin d'une ligne, ils empêchent le curseur d'aller à la ligne. Leur emploi entraîne l'adjonction d'un espace à gauche et à droite des articles numériques.
PRINT @	Imprime des données à un endroit précis de l'écran. Deux coordonnées, l'une pour la rangée, l'autre pour la colonne, doivent être fournies : ex. : PRINT @ (4,5), "HELLO" fait apparaître HELLO sur la rangée 4, à compter de la colonne 5. Le format de l'écran est de 16 rangées sur 32 colonnes ; la numérotation débute à la rangée 0 et à la colonne 0.
READ	Associe des valeurs à des variables en utilisant des séries d'articles placés à la suite de DATA.
REM	Précède une remarque destinée au programmeur ou à l'utilisateur. Le contenu des REMarques n'est pas pris en compte par l'ordinateur.
RETURN	Renvoie au programme principal après l'exécution d'un sous-programme.
RUN	Provoque l'exécution du programme en cours.
SAVE	Sauvegarde un programme sur cassette (pas étudiée en détail).

STEP	Détermine le pas utilisé pour les compteurs FOR... NEXT; par exemple : FOR X = 1 TO 10, STEP 0,5. Si STEP est omis, il faut sous-entendre STEP 1.

Fonctions et opérateurs

Opérateurs arithmétiques

+	addition, ex. : 1 + 2
−	soustraction, ex. : 3 − 4
/	division, ex. : 3/4
*	multiplication, ex. : 3 * 4

L'ordre d'exécution n'est pas approfondi dans cet ouvrage. Dans un souci de clarté, un large recours est fait aux parenthèses pour les expressions assez complexes.

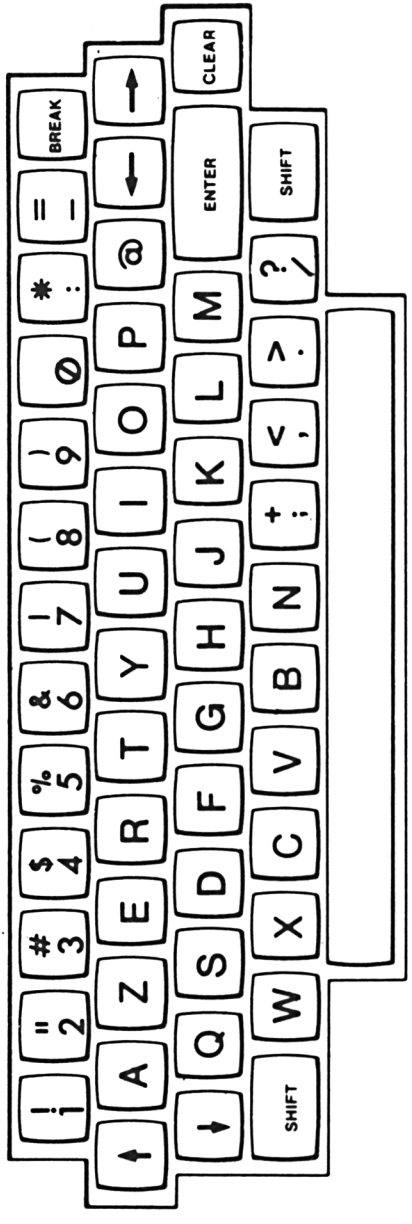
Opérateurs logiques

=	égal à
<	plus petit que
<=	inférieur ou égal à
>	plus grand que
>=	supérieur ou égal à
<>	différent de

Fonctions

CHR$	Produit le caractère dont le code suit; ex. : CHR$(135). Utilisé ici seulement dans le but de produire des caractères graphiques. Voir liste des codes p. 51.
COS	Donne le cosinus du nombre qui suit; ex. : COS(55).
INT	Arrondit au nombre entier inférieur; ex. : INT(5,5) donne 5.
RND	Produit un nombre aléatoire compris entre 0 et 1. Afin d'obtenir un nombre entier aléatoire, il convient d'utiliser l'expression suivante : INT (RND*(n)) (n déterminant l'éventail des nombres aléatoires produits); ex. : INT(RND*5).
SQR	Donne la racine carrée : ex. : SQR(25). Mentionné brièvement dans l'ouvrage.

Le clavier de notre ordinateur



Annexe 3 Lectures et ouvrages suggérés

Les revues

Parmi les dizaines de publications françaises, nous avons sélectionné les magazines généraux de micro-informatique qui publient des programmes et des commentaires accessibles au débutant.

L'ordinateur individuel une revue mensuelle de bon niveau, parfois très technique ;
Micro 7 un magazine mensuel toujours intéressant ;
Science et Vie Micro une revue générale d'accès très facile et de très bonne qualité.

Nous vous recommandons également les magazines spécialisés, s'il en existe un pour votre micro-ordinateur. Ajoutons enfin les encyclopédies par fascicules, à parution hebdomadaire, qui offrent une approche didactique et progressive : *ABC Informatique* et *BASIC PLUS*.

Les livres

Les livres suivants, sélectionnés parmi une bibliographie très abondante, que vous pouvez consulter dans Le Tout Micro (Hachette informatique), vous permettront d'approfondir vos connaissances en Basic.

Basic pour tous S. Brebion, J. Boisgontier (PSI)
Le Basic bien programmé A.P. Stephenson (Dunod)
Le Manuel du Basic M. Waite, M. Pardee (Hachette informatique)
Jeux, graphiques et sons S. Curran, R. Curnow (Hachette informatique)

Corrigé des exercices

Chapitre 1

1) PRINT 56.4 + 76.3 (E)

2) PRINT 74/13.1 (E)

3) PRINT 45*7 (E)

4) PRINT 1025 – 96 (E)

5) PRINT "
SUSAN"

(Il y a 27 espaces avant "SUSAN", ce qui permet de l'imprimer sur une nouvelle ligne de l'écran.)

6) PRINT "R",,,,"A",,,,"Y"

7) PRINT "ROUGE ";"ORANGE ";"BLEU ";"ROSE"

8) Voici notre chien :

```
OO-----/----
OO----/
1   1
```

Et voici notre programme :

```
CLS:PRINT@(0,16),"OO-----/----",
,,"OO----/",,,"  1   1"
```

9) CLS :PRINT@(8,12),"CAROLINE"

Nous avons utilisé un mot de huit lettres, aussi nous laissons 12 espaces (une rangée en contient 32) de chaque côté de celui-ci.

10) CLS :PRINT@(0,31),"/" :PRINT@(1,30
),"/" :PRINT@(2,29),"/"..... etc.

... etc.

Chapitre 2

1) Non, celui-ci ne marchera pas, pourquoi ? En raison des deux-points placés après PRINT qui marquent la fin de l'instruction. La deuxième instruction de la ligne 10 (« Salut la compagnie ») est « grammaticalement » incorrecte.

2) Celui-ci ne marchera pas non plus. L'instruction END précède PRINT dans l'ordre numérique des lignes (le seul qui compte).

3) Celui-ci fonctionnera. Voici ce que l'on obtient :

```
000 000 000
000 000 000    (Trois fleurs très stylisées).
 1   1   1
 1/  1/  1/
```

4) Celui-ci fonctionnera également. Les instructions PRINT vides déplacent simplement le curseur d'une ligne vers le bas, aussi vous obtenez :

```
SALUT


AU REVOIR
```

Chapitre 3

1) a) impossible : A est un nom de variable numérique, et "POMMES" est une chaîne ;
   b) oui, c'est possible !
   c) impossible. On peut placer 56 dans une chaîne, mais il faut mettre les guillemets (" ").

2)
```
10 REM PONT
15 CLS
20 LET A$ = CHR$(128)+CHR$(128)+CHR$(128)
30 LET B$ = CHR$(128)+CHR$(143)+CHR$(128)
40 PRINT A$ :PRINT A$ :PRINT B$ :PRINT B$
50 END
```

3)
```
10 REM NOM
15 CLS
20 LET A$ = "JEAN DUPONT"
30 PRINT@ (0,5),  A$
40 PRINT@ (8,5),  A$
50 PRINT@ (15,5),  A$
60 END
```

4)
```
10 REM PROGRAMME X ET Y
20 INPUT "DONNEZ-MOI DEUX NOMBRES";X,Y
30 PRINT X;"+";Y;"=";X+Y
40 PRINT X;"-";Y;"=";X-Y
50 PRINT X;"*";Y;"=";X*Y
60 PRINT X;"/";Y;"=";X/Y
70 END
```

5)
```
10 REM *** DESSINEZ-LE VOUS-MÊME" ***
20 PRINT "DONNEZ-MOI UNE FORME A DESSINER"
30 PRINT "A L'AIDE DE 3 CODES CHR$"
40 PRINT "EX. : 128, 143, 137"
50 PRINT "ET QUAND JE L'AURAI DESSINÉE"
60 PRINT "DONNEZ-M'EN UNE SECONDE"
```

```
70 INPUT X, Y, Z
80 PRINT CHR$(X)+CHR$(Y)+CHR$(Z)
90 GOTO 70
```

Pas d'instruction END, cette fois, car le programme est pris dans une boucle qui ne s'ouvre que lorsque l'on appuie sur BREAK. Gardez à l'esprit le fait que tous ces programmes ne sont que des solutions possibles parmi d'autres. La vôtre peut être exacte, tout en étant légèrement différente de la nôtre.

6)
```
10 PRINT 17 + INT(RND*10)
20 GOTO 10
```

7) Voici notre liste de variables :

Dessin du camion	C$
Position de colonne	N
Vitesse de déplacement	D

et notre programme :

```
10 REM CAMION
20 LET C$ = CHR$(128)+CHR$(128)+CHR$(136)
30 LET N = 0
40 LET D = 1 + INT(RND*3)
50 PRINT@ (8,N), C$
60 LET N = N + D
70 CLS
80 GOTO 50
```

Chapitre 4

1) D'abord, une liste de variables :

Le nombre aléatoire produit par l'ordinateur	N
Votre nombre	M

Le programme, maintenant :

```
10 REM DEVINEZ LE NOMBRE
20 LET N = 1 + INT(RND*6)
30 PRINT "JE PENSE A UN NOMBRE"
40 PRINT "COMPRIS ENTRE 1 et 6"
50 PRINT "DEVINEZ CE QUE C'EST"
60 INPUT M
70 IF M = N THEN GOTO 100
80 PRINT "NON, ESSAYEZ ENCORE"
90 GOTO 60
100 PRINT "BRAVO! C'ÉTAIT BIEN";N
110 END
```

2) Une seule variable, cette fois :

Nombre choisi	N

Le programme, maintenant :

```
10 REM***GÉNÉRATEUR DE MESSAGES***
20 PRINT "CHOISISSEZ UN NOMBRE"
```

```
30 PRINT "COMPRIS ENTRE 1 ET 6"
40 PRINT "ET J'AFFICHERAI UN MESSAGE"
50 INPUT N
60 IF N = 1 THEN PRINT "BONJOUR!"
70 IF N = 2 THEN PRINT "REBONJOUR! BEAU TEMPS, NON?"
80 IF N = 3 THEN PRINT "SALUT! ET AU BOULOT!"
90 IF N = 4 THEN PRINT "ET SI ON PARTAIT EN VACANCES?"
100 IF N = 5 THEN PRINT "BONJOUR. ET CES VACANCES?"
110 IF N = 6 THEN PRINT "BONJOUR, VOUS!"
200 END
```

3)

```
10 REM LE CHARMEUR DE SERPENT 2
20 CLS
30 LET N = 15
40 PRINT@ (N,10), "SSSSS";
50 LET N = N - 1
60 IF N = - 1 THEN GOTO 80
70 GOTO 40
80 GOTO 80
```

4)

```
10 REM ***AMBULANCE***
15 CLS
20 LET A$ = CHR$(128)+CHR$(128)+CHR$(128)+CHR$(128)
30 LET B$ = CHR$(128)
40 LET C$ = "URGENCE"
50 PRINT@ (8,6), A$+A$+B$+B$;
60 PRINT@ (9,4), A$+A$+A$;
70 PRINT@ (10,4), A$+A$+A$;
80 PRINT@ (11,5), B$:PRINT@ (11,14), B$;
90 PRINT@ (9,6), C$;
100 FOR D = 0 TO 50: NEXT D
110 PRINT@ (9,6), A$+A$+B$;
120 FOR D = 0 TO 25: NEXT D
130 GOTO 90
```

5)

```
10 REM CARACTERES GRAPHIQUES
20 FOR G = 128 TO 143
30 PRINT CHR$(G);" ";
40 NEXT G
50 END
```

6)

```
10 REM ***CHRONOMETRE***
20 PRINT "DONNEZ-MOI UN NOMBRE DE SECONDES"
30 PRINT "A COMPTER ET JE LES"
40 PRINT "COMPTERAI POUR VOUS"
50 INPUT S
```

```
60  FOR N = 0 TO S - 1
70  PRINT N
80  FOR D = 0 TO 460: NEXT D
90  NEXT N
100 PRINT S; "LE TEMPS EST ECOULE!"
110 END
```

Chapitre 6

1)
```
10  REM PROGRAMME DES QUATRE NOMBRES
20  DIM A(4)
30  PRINT "DONNEZ-MOI QUATRE NOMBRES"
40  PRINT "ET JE LES AFFICHERAI"
50  FOR N = 1 TO 4
60  INPUT "NOMBRE ";A(N)
70  NEXT N
80  FOR N = 1 TO 4
90  PRINT,,A(N)
100 NEXT N
110 END
```

2)
```
10  REM IMPRESSION D'UN MOTIF
20  DIM C(20) : LET N = 1
30  PRINT "DONNEZ-MOI 20 CODES CHR$"
40  PRINT "ET J'IMPRIMERAI LES
45  CARACTERES CORRESPONDANT"
50  PRINT "ENTREZ 0 QUAND VOUS AUREZ
55  FINI"
60  INPUT "CODE";C(N)
70  IF C(N) = 0 THEN GOTO 200
80  LET N = N + 1
90  IF N = 21 THEN GOTO 200
100 GOTO 60
200 REM DESSIN DES CARACTERES
210 CLS
220 FOR M = 1 TO (N-1)
230 PRINT @ (8,6+M), CHR$(C(M));
240 NEXT M
250 END
```

3)
```
10  REM LISTE DES NOMS INVERSES
20  DIM N$(5)
30  PRINT "DONNEZ-MOI 5 NOMS"
40  PRINT "ET J'EN INVERSERAI L'ORDRE"
50  FOR X = 1 TO 5
60  INPUT "NOM";N$(X)
70  NEXT X
80  FOR X = 5 TO 1 STEP -1
90  PRINT N$(X)
100 NEXT X
110 END
```

4)
```
10  REM SERPENT
20  CLS
30  REM DONNEES POUR LE CORPS
40  DIM R(19): DIM C(19)
50  FOR X = 1 TO 19
60  READ R(X): READ C(X)
70  NEXT X
80  DATA 9,3,8,4,9,5,9,6,10,7,10,8,9,
    9,9,10
90  DATA 8,11,8,12,9,13,10,14,10,15,
    9,16,9,17
100 DATA 8,18,7,19,6,20,4,20
110 REM DESSIN DU CORPS
120 FOR P = 1 TO 19
130 PRINT @ (R(P),C(P)), CHR$(128);
140 PRINT @ (R(P)+1,C(P)), CHR$(128);
150 NEXT P
160 REM DESSIN DE LA LANGUE
170 PRINT @ (3,20), "1";: PRINT @
    (2,20), "1";
180 GOTO 180
```

5) Nous utiliserons le tableau N$ pour les noms, le tableau A$ pour les adresses et le tableau M$ pour la mise en correspondance des deux. Nous n'avons placé que trois noms et trois adresses dans le fichier et non pas dix comme vous l'aurez sans doute fait.

```
10  REM FICHIER DES NOMS ET
    ADRESSES
20  DIM N$(3): DIM A$(3): LET C = 0
30  REM LECTURE DES DONNEES DE FICHIER
40  FOR X = 1 TO 3
50  READ N$(X): READ A$(X)
60  NEXT X
70  DATA "RUFUS DUPONT","3 RUE VINCI"
80  DATA "LUC BEL","80 AV. GAMBETTA"
90  DATA "SIMONE PASSY","13 BD DUPONT"
100 REM DEMANDE DE L'ADRESSE
110 PRINT "NOM DE LA PERSONNE"
120 INPUT "(PRENOM, NOM DE FAMILLE)";M$
130 FOR X = 1 TO 3
140 IF M$ = N$(X) THEN PRINT A$(X): LET
    C = 1
150 NEXT X
160 IF C = 0 THEN PRINT "NE FIGURE PAS
    AU FICHIER"
170 INPUT "UN AUTRE ESSAI ? O/N";R$
180 IF R$ = "O" THEN LET C = 0: GOTO
    100
190 END
```

Index

Imprimé en France par Offset-Aubin, 86000 Poitiers
Dépôt légal n° 9741-11-1984 — Imprimeur n° P 13161
Collection n° 20 — Edition n° 01
10/0005/8
ISBN 2.01.010301.7

PREMIERS PAS EN BASIC

Cet ouvrage s'adresse à tous ceux et à toutes celles qui veulent apprendre le langage le plus utilisé sur tous les micro-ordinateurs familiaux : le Basic.

Des jeux et des exercices gradués leur feront découvrir les principes de base de la programmation.

L'auteur, animé d'un réel souci pédagogique présente ici une véritable méthode d'apprentissage claire, pratique et simple.

Précisons qu'il n'est même pas nécessaire de posséder un micro-ordinateur, ni d'être fort en maths !

L'informatique, c'est un jeu d'enfants !

ACCES

99.00/84/11

9 782010 103018

10/0005/8

Imprimé en France

SUD OFFSET - 94 RUNGIS

HACHETTE Informatique

Document numérisé avec amour par :

AMSTRAD CPC

MÉMOIRE ÉCRITE

ACME

https://acpc.me/