P.S.I.

LA DÉCOUVERTE DE L'AMSTRAD

CPC 464 ET 664/6128

DANIEL-JEAN DAVID

LA DÉCOUVERTE DE L'AMSTRAD
CPC 464 ET 664/6128

Autres ouvrages relatifs à l'Amstrad

— Exercices en BASIC pour Amstrad - Maurice Charbit
— BASIC Amstrad - 1. Méthodes pratiques - Jacques Boisgontier et Bruno Césard
- 2. Programmes (à paraître) - Jacques Boisgontier
— BASIC plus 80 routines sur Amstrad - Michel Martin
— Assembleur de l'Amstrad - Marcel Henrot
— 102 programmes pour Amstrad - Jacques Deconchat
— Super jeux Amstrad - Jean-François Sehan
— Amstrad en famille - Jean-François Sehan
— Clefs pour Amstrad - Daniel Martin

Ouvrages d'initiation à l'informatique

— Visa pour l'informatique - **Jean-Michel Jégo**
— Visa pour le BASIC - **Jean-Michel Jégo**
— Le logotron informatique - Jean-Pierre Petit
— Les mots de la micro - **Alan Freedman**

Pour tout problème rencontré dans les ouvrages P.S.I.
vous pouvez nous contacter au numéro ci-dessous :

Numéro Vert/Appel Gratuit en France

16 (05) 21 22 01

(Composer tous les chiffres, même en région parisienne)

La loi du 11 mars 1957 n'autorisant, aux termes des aliénas 2 et 3 de l'article 41, d'une part, que les «copies ou reproductions strictement réservées à l'usage privé du copiste et non destinées à une utilisation collective», et, d'autre part, que les analyses et les courtes citations dans un but d'exemple et d'illustration, «toute représentation ou reproduction intégrale, ou partielle, faite sans le consentement de l'auteur ou de ses ayants droit ou ayants cause, est illicite» (aliéna 1er de l'article 40)

Cette représentation ou reproduction, par quelque procédé que ce soit, constituerait donc une contrefaçon sanctionnée par les articles 425 et suivants du Code Pénal

© Éditions du P.S.I. - B.P. 86 - 77402 Lagny cedex
1985
ISBN: 2-86595-269-1

DANIEL-JEAN DAVID

LA DÉCOUVERTE DE L'AMSTRAD

CPC 464 ET 664/6128

Éditions du P.S.I.
1985

Daniel-Jean David enseigne l'informatique de gestion à l'Université de Paris-1 Panthéon-Sorbonne.

Par ailleurs il enseigne l'utilisation des microprocesseurs à l'E.N.S.A.M.

Ses sujets de recherche vont de l'informatique graphique aux techniques d'interface des microprocesseurs et des systèmes multi-processeurs.

Spécialiste des microprocesseurs, il a donné à Paris de nombreux séminaires sur les microprocesseurs et les langages de programmation des micro-ordinateurs. Il a écrit de nombreux articles dans les revues spécialisées.

SOMMAIRE

PRÉSENTATION

Ce livre a pour but de vous permettre de tirer le meilleur parti possible de votre micro-ordinateur Amstrad. Après une introduction formée de rappels généraux sur l'informatique, il comprend essentiellement une introduction progressive au langage BASIC qui est le langage de programmation utilisé le plus souvent sur Amstrad.

Bien entendu, on y exploite au maximum les particularités de l'Amstrad. La structure de cette partie est conçue pour permettre l'acquisition progressive des connaissances : elle est formée de chapitres, ou plutôt, de séries.

Dans chaque série, on bâtit petit à petit un programme, par variations continues, en introduisant peu à peu les notions nouvelles.

Il est recommandé au lecteur de bien suivre cette progression, d'essayer réellement sur son micro-ordinateur les différentes versions des programmes et même, d'en imaginer d'autres.

C'est la condition nécessaire d'acquisition de connaissances durables. Toutefois, ce livre devrait aussi permettre aux personnes qui n'ont pas encore de micro-ordinateur de se faire une idée des possibilités de l'Amstrad.

Enfin, l'ouvrage se termine par des annexes où sont fournies des informations de référence : explication de chaque instruction BASIC, messages d'erreur, points particuliers traités sous forme de questions et réponses.

Dernière précision : tous les programmes cités ont été réellement essayés au clavier d'un Amstrad.

Le livre s'applique aux trois modèles d'Amstrad actuellement sur le marché : le CPC 464 et le CPC 664/6128. Les différences (peu nombreuses) sont signalées. La plus fondamentale est que le 664/6128 possède une unité de disquettes.

Le BASIC de l'Amstrad est probablement le plus riche qui soit disponible sur le marché des micro-ordinateurs domestiques. Nous ne pouvons, dans ce livre d'introduction, en traiter tous les aspects. Ce livre traite de tout ce qui est fondamental en BASIC et il aborde les problèmes de graphismes et sons afin de permettre au lecteur de profiter de ces éléments qui sont parmi les plus attrayants que peut offrir la micro-informatique. Mais, par manque de place, il est obligé de laisser de côté de nombreuses instructions.

CHAPITRE 1
PRISE DE CONTACT

BITS-OCTETS-INFORMATIONS

A peine êtes-vous assis devant votre Amstrad et l'avez-vous mis sous tension qu'il affiche un message rappelant le nom du constructeur et indiquant que la machine a 64 K (c'est la taille mémoire en *octets*) et qu'elle est prête (Ready) à travailler. Que sont donc ces *octets* dont on vous annonce qu'il y en a 64 K (et 128 K pour le 6128) ?

Pour le comprendre - et nous nous en excusons - il nous faut voir un certain nombre de notions fondamentales. Nous essaierons d'être brefs. L'Amstrad est un ordinateur, c'est-à-dire une machine de traitement automatique des informations. On conçoit que les ordinateurs aient un champ d'application extrêmement vaste puisque, en définitive, toute activité humaine se ramène à un certain traitement d'informations.

Une des opérations essentielles que l'ordinateur doit pouvoir effectuer sur une information est de la mémoriser, pour être capable de l'utiliser à différents moments. Mais la mémorisation de l'information implique sa mise sous une forme physique convenable pour pouvoir être stockée dans les circuits de l'ordinateur.

Dans l'état actuel de la technologie, le seul codage pratique est de type binaire, car on sait très bien réaliser des éléments capables de prendre deux états bien distincts, par exemple : présence ou absence d'un trou sur une carte, élément de bande magnétique aimanté dans un sens ou dans l'autre, élément de circuit électronique porté au potentiel 5 V, ou restant à 0, etc.

Un tel élément qui, en tant qu'information est en somme capable de contenir la réponse par oui ou par non à une certaine question s'appelle un *bit* (abréviation anglaise de chiffre binaire : Binary digIT).

Mais un seul bit forme le plus souvent une information trop élémentaire à manipuler de façon pratique. C'est pourquoi on manipule généralement des groupes de bits. Le groupe le plus souvent envisagé est le groupe de 8 bits ou octet. Pour comprendre ce que signifie "en avoir 64 K, c'est-à-dire environ 64 000", il nous faut maintenant voir ce que l'on peut repésenter avec un octet, c'est-à-dire quelles informations peuvent être enfermées dans un octet.

Pour visualiser un octet sur le papier, il nous faut introduire deux symboles correspondant aux deux états que peut prendre chaque bit de l'octet. Si nous prenons pour symboles 0 et 1, un octet pourra être, par exemple, 010000001 ou 10100101 ou encore 01101001. Notons tout de suite que, comme l'on a deux possibilités pour chaque bit, on a $2^8 = 256$ possibilités différentes pour un octet.

Maintenant, si nous voulons qu'un octet représente un nombre, c'est très facile. Il suffit de considérer que l'on a exprimé le nombre dans le système de numération binaire (à base 2). Ainsi, par exemple, 01000001 vaudra $1 + 0*2 + 0*2^2 + 0*2^3 + 0*2^4 + 0*2^5 + 1*2^6 + 0*2^7 = 1 + 64 = 65$, de la même façon qu'en décimal 1702 vaut $2 + 0*10 + 7*10^2 + 1*10^3$. Chaque bit représente un chiffre du système binaire, d'où son nom. Le nombre le plus petit que l'on puisse représenter est 00000000 (0) ; le plus grand est 11111111 (255) : on retrouve les 256 combinaisons. On voit aussi une chose : comme l'on veut pouvoir manipuler des nombres plus grands que 255, il faudra quelquefois que les nombres occupent plusieurs octets.

Peut-on ranger autre chose que des nombres dans un octet ? Bien sûr ! Supposons qu'on décide que 01000001 = A, 01000010 = B, etc. pour toutes les lettres. Là encore, on peut avoir un jeu de 256 caractères différents, ce qui permet les lettres majuscules et minuscules, les chiffres, les caractères de ponctuation et bien d'autres. On pourra alors stocker n'importe quel texte dans la mémoire à raison d'un caractère par octet.

On peut donc ranger un texte de 64 000 caractères dans la mémoire d'un Amstrad. Cela nous donne maintenant une meilleure idée de ce que nous avons comme mémoire.

Mais il y a encore une autre catégorie d'informations qui doivent entrer dans la mémoire, et c'est même une caractéristique fondamentale des ordinateurs. En effet, pour fonctionner, un ordinateur a besoin d'instructions qui lui disent ce qu'il a à faire. Ces instructions résident en mémoire, au même titre que les informations à traiter ; de façon analogue, un employé qui effectue des calculs de comptabilité n'a-t-il pas en mémoire la liste des opérations qu'il doit effectuer à côté des chiffres qu'il doit manipuler ?

LA CONFIGURATION DE L'AMSTRAD

La mémoire n'est qu'une partie de la configuration dont on dispose avec un Amstrad. Sur les gros ordinateurs, on distingue facilement, car ils occupent chacuèn une armoire, les éléments principaux qui sont l'unité centrale (où s'effectuent les traitements) et les périphériques qui servent à communiquer avec le monde exstérieur (notamment saisir les données à traiter et fournir les résultats obtenus).

Les mêmes éléments existent sur un Amstrad. Extérieurement, on ne voit que les périphériques :

– Le clavier : qui va nous servir à entrer des données et des instructions.

– L'écran de télévision : sur lequel s'afficheront les résultats sur 25 lignes de 40 caractères (il y a d'autres modes d'affichage où la largeur de ligne est de 20 à 80 caractères ; elle est de 40 nà la mise sous tension). Le couple clavier/écran est l'instrument du dialogue entre vous et votre machine.

– L'unité de cassettes magnétiques : à la différence des périphériques de communication précédents, il s'agit plutôt d'un périphérique de stockage ou mémoire de masse qui permet de stocker des programmes ou des données si la mémoire centrale est insuffisante. Mais la cassette est aussi un périphérique de communication entre Amstrad : vous pouvez envoyer un programme ou des données à un ami, qui possède aussi un Amstrad.

Sur le 664/6128, à la place de l'unité de cassettes du 464, c'est une unité de disquettes que vous avez. Elle permet tout autant et même mieux (plus rapidement), les sauvegardes de programmes. De toutes façons, vous pouvez connecter un magnétocassette à un 664/6128 et le 464 peut recevoir une unité de disquettes externe.

L'unité centrale est cachée, mais elle existe et elle est en fait formée de deux éléments : le processeur et la mémoire centrale.

– Le processeur est ici un microprocesseur, c'est-à-dire un circuit intégré à grande échelle capable, à lui seul, de commander tout le système : il cherche en mémoire les instructions successives, les interprète et les exécute ; il envoie aux autres composants du système les ordres nécessaires. Dans le cas de l'Amstrad, le microprocesseur est un Z-80, l'un des plus efficaces du marché.

– La mémoire. Les 64 000 octets vus précédemment n'en sont qu'une partie. C'est en fait 96 000 octets qui sont présents (les informaticiens disent 96 K où K = 2^{10} = 1024). Parmi les 96 K, 32 K sont de la ROM (Read Only Memory), mémoire écrite une fois pour toutes, encore appelée Mémoire Morte ou MEM, qui contient les programmes invariables permettant à l'Amstrad de se mettre au service de l'utilisateur (c'est le système d'exploitation que nous présentons plus en détail à la section suivante). Les 64 K restants sont de la RAM (Random Access Memory), c'est-à-dire de la mémoire que l'on peut lire ou écrire (ou Mémoire Vive MEV). Elle contient les programmes de l'utilisateur et les don-

nées variables. Seuls 48 K en sont disponibles et l'Amstrad en réserve 6 pour son usage, d'où 42 K restant libres pour l'utilisateur. Les 16 autres K de RAM que nous avons dits indisponibles sont réservés pour la mémoire d'écran.

La figure 1 donne une vue synoptique de la configuration d'un Amstrad. Il nous reste à expliquer la ligne "BASIC 1.0" ou "BASIC 1.1" (sur 664). Ce sera le but de la section suivante.

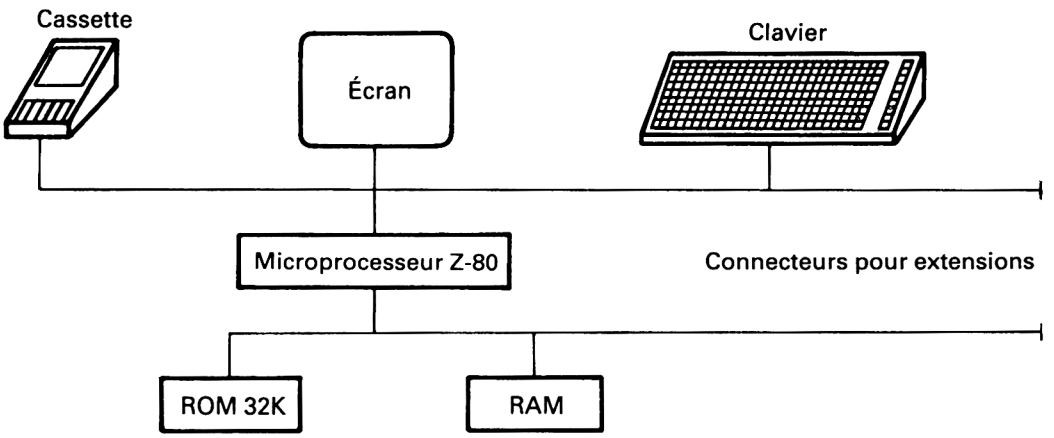


Figure 1.1. – Synoptique d'un Amstrad

PROGRAMMES SYSTÈME D'EXPLOITATION - BASIC

On vient de voir que le microprocesseur, pièce maîtresse de l'unité centrale était capable de chercher en mémoire les instructions successives, de les décoder et d'y obéir. De fait, il n'est capable que de cela ! Ensuite, pour obtenir quoi que ce soit de l'ordinateur, il faudra lui fournir les instructions convenables. Sans instruction précise, l'ordinateur ne sait rien faire, il n'a aucune initiative.

Une suite d'instructions que l'ordinateur doit exécuter successivement s'appelle un programme. Fournir une telle suite d'instructions permettant de résoudre un certain problème s'appelle programmer.

Vous concevez donc que vous allez devoir fournir des programmes à votre Amstrad, que vous introduirez par le clavier. Mais, pour que l'Amstrad prenne en compte ce que vous frappez au clavier, il faut qu'un programme le lui ordonne. Il n'aurait pas tout seul l'initiative d'aller voir si quelqu'un tape sur le clavier...

Heureusement, nous n'avons pas à écrire ce programme. En effet, l'Amstrad - comme d'ailleurs tout autre ordinateur - est livré avec un ensemble de programmes fondamentaux qui sont indispensables à son utilisation : programme qui lit le clavier, programme qui affiche sur l'écran, programme qui gère les cassettes, programme qui attend les instructions de l'utilisateur, etc. L'ensemble de ces programmes s'appelle le système d'exploitation.

Ces programmes résident en ROM, afin d'être conservés en permanence pour être disponibles dès qu'on met l'Amstrad sous tension. Le message qui s'affiche sur l'écran dès cette mise sous tension est une manière de dire à l'utilisateur que le système d'exploitation se met à sa disposition et attend ses ordres.

Une composante très importante du système d'exploitation de l'Amstrad est l'interpréteur BASIC qui va nous permettre de fournir des instructions à l'Amstrad sous une forme commode pour nous.

La question se pose en effet de savoir sous quelle forme - ou en quel langage - nous devons fournir nos instructions à l'Amstrad. A dire vrai, un ordinateur ne "comprend" qu'un seul langage, le langage-machine ou binaire.

Par exemple, en langage-machine de l'Amstrad, "ajouter 2 et 3" se dirait 00111110 00000010 11000110 00000011 00110010 01010000 01010000 01010000... C'est extrêmement compliqué à comprendre et à utiliser pour l'homme ! C'est pourquoi d'autres langages de programmation ont été inventés, qui s'appellent langages évolués et sont plus proches des notations mathématiques usuelles, et donc plus faciles à utiliser. Dans un tel langage, "ajouter 2 et 3" se dirait, par exemple, A=2+3; c'est beaucoup plus compréhensible, n'est-ce pas ?

BASIC est l'un de ces langages évolués, le plus répandu à l'heure actuelle sur les micro-ordinateurs. C'est probablement le plus simple à utiliser pour les débutants et son nom est l'abréviation de Beginners' All-purpose Symbolic Instruction Code : codage symbolique d'instructions d'usage général pour les débutants.

C'est lui que nous utiliserons sur Amstrad, mais nous ne pouvons le faire sans l'aide d'un programme du système d'exploitation. En effet, l'Amstrad ne comprend vraiment que son langage-machine ; il faut donc un programme du système d'exploitation qui prenne chaque instruction de notre programme BASIC et la traduise en binaire pour l'exécuter : c'est le rôle - essentiel - joué par l'interpréteur BASIC.

En résumé, l'affichage qui apparaît sur l'écran à la mise en route signifie : l'interpréteur BASIC du système d'exploitation est à votre disposition. Le dernier mot Ready qui réapparaîtra à chaque fois que l'Amstrad aura terminé une commande signifie "Je suis prêt et j'attends que vous tapiez la prochaine instruction".

Faisons-le...

LES DEUX MODES DE FONCTIONNEMENT DE BASIC

Mettons-nous au clavier et tapons :

BONJOUR AMSTRAD touche **ENTER**

(Tout message tapé par l'utilisateur se termine en principe par cette touche **ENTER** - entrée -. Dans la suite, nous cesserons progressivement de la faire figurer : elle sera sous-entendue et devra être tapée.)

L'ordinateur répond :

```
Syntax Error
Ready
```

En effet, bien que notre message soit très poli, il ne forme pas une instruction BASIC correcte, et par suite, l'Amstrad (ou plutôt l'interpréteur BASIC) la refuse. (*N.B.*– En principe, votre message sera tapé en minuscules ; cela n'a pas d'importance : des majuscules auraient donné le mêmez résultat.)

Il ne faut pas faire de complexes à propos des messages d'erreur, mais chercher calmement l'erreur que l'on a faite ; le comportement de la machine est, dans tous les cas, parfaitement rationnel. L'Amstrad a tout un répertoire de messages d'erreur qui n'ont pas d'autre but que celui de nous aider à les corriger.

Tapons maintenant :

?"Bonjour! Je suis un Amstrad" **ENTER**

(Les espaces s'obtiennent avec la barre d'espace, les lettres peuvent être tapées au choix en majuscules ou en minuscules.)

Cette fois, on a tapé une bonne instruction et l'on obtient comme réponse :

```
Bonjour! Je suis un Amstrad
Ready
```

Vous n'obtenez pas cette réponse ? Êtes-vous certain d'avoir bien tapé tous les caractères y compris les guillemets ?

Le point d'interrogation signifie simplement "Imprimer". On constate que l'on a obtenu une réponse immédiate à l'instruction. De même, si l'on tape :

?2+2 **ENTER**

l'Amstrad fera le calcul et imprimera immédiatement le résultat. On a donc un mode de fonctionnement à peu près semblable à celui d'une calculatrice de poche, où une instruction est exécutée dès qu'elle vient d'être tapée. On dit qu'il s'agit du "mode immédiat" ou encore "mode direct".

L'Amstrad a un second mode de fonctionnement. Tapons :

20 ?2+2 **ENTER**

Rien ne se produit. Tapons encore :

10 ?"Bonjour" **ENTER**

Toujours rien. Tapons :

RUN (les lettres R U N suivies de **ENTER**).

Cette fois, on obtient sur l'écran :

```
Bonjour
 4
Ready
```

Que s'est-il passé ? Eh bien, on a fonctionné dans le second mode où les instructions ne sont pas exécutées immédiatement, mais mises en mémoire pour exécution ultérieure. Elles forment alors un programme qui est exécuté lorsqu'on tape la commande RUN en mode direct.

Sauf une ou deux exceptions, ce sont exactement les mêmes instructions qui peuvent être données en mode direct ou dans le second mode, qui s'appelle "mode différé" ou "mode programmé". Alors, à quoi reconnaît-on le mode ?

C'est très simple : en mode différé, toute instruction possède en tête un numéro de ligne, alors qu'en mode direct, il n'y a pas de numéro de ligne :

```
?2+2       mode direct;
10 ?2+2    mode différé (il faut taper RUN pour avoir la réponse).
```

En plus d'imposer le mode programmé, le numéro de ligne joue un autre rôle : on voit sur l'exemple précédent que l'instruction 10 à été exécutée avant l'instruction 20, bien qu'elle ait été tapée après : les instructions sont exécutées non pas dans l'ordre chronologique où elles ont été tapées, mais par ordre de numéros de ligne croissants.

Les Amstrad sont naturellement beaucoup plus utilisés en mode programmé, mais, avant, pour nous familiariser, nous allons effectuer quelques calculs arithmétiques en mode direct.

ARITHMÉTIQUE EN MODE DIRECT

Comme toute calculatrice, un Amstrad permet d'évaluer des expressions plus compliquées que 2+2 ! Dans tous les cas, on doit commencer par un ? ou par le mot PRINT dont il est l'abréviation : cela signifie "Imprimez le résultat de l'expression qui suit".

Essayez (les **ENTER** sont sous-entendus) :

?5−3.5	(soustraction;	résultat 1.5)
?3*12	(multiplication;	résultat 36)
?1/3	(division;	résultat .333333333)
?2 ↑ 5	(élévation à la puissance;	$2^5 = 32$)
	(il est listé ^ sur certaines imprimantes)	

On voit donc quels sont les signes d'opération fondamentaux; Noter * et non × pour la multiplication, ↑ pour l'élévation à la puissance (on ne peut, au clavier, mettre un nombre plus haut que l'autre).

VARIABLES

Étant donné une expression, on peut en faire autre chose que d'imprimer sa valeur : on peut mettre la valeur en mémoire pour utilisation ultérieure dans une autre expression.

Pour cela, il suffit de donner un nom à l'expression, en tapant par exemple :

A=2/3 (ou a=2/3)

On dit qu'on a constitué une variable de nom A. L'Amstrad lui attribue automatiquement un emplacement mémoire (que vous n'avez pas à connaître : l'Amstrad se charge entièrement de la gestion de la mémoire). Ensuite, le résultat est calculé et rangé dans la case mémoire considérée ; il n'apparaît pas sur l'écran.

La variable peut maintenant être utilisée dans une autre expression ; si l'on tape :

?2*A

on obtient :

1.333333333

Quels noms de variables peut-on prendre ? Les noms de variables sont pratiquement arbitraires, à ceci près qu'ils doivent se plier aux contraintes suivantes :

- premier caractère : lettre ;
- caractères suivants : lettres ou chiffres (*ex.* A, A1, VAR, RESULT, H2SO4).

L'Amstrad permet jusqu'à 40 caractères, mais le bon sens vous conseillera de vous limiter à beaucoup moins, par exemple 5 ou 6.

Le fait de pouvoir utiliser plusieurs paramètres permet de choisir des noms *parlants* comme RESULT, TAUX, CAP..., mais il est déconseillé que le nom contienne un des mots particuliers au langage BASIC qu'on appelle les mots-clés : CHIFFRE est déconseillé, car il contient IF qui est un mot ayant une signification particulière pour BASIC. En tout cas, un nom de variable n'a pas le droit d'être identique à un mot-clé.

La liste des mots-clés "réservés" est fourni en annexe.

En principe, les mots-clés doivent être séparés du reste par des espaces, ne serait-ce que pour la lisibilité.

L'usage d'une variable ne détruit pas sa valeur, ou plutôt n'efface pas la mémoire correspondante. Ce n'est que lors d'une nouvelle instruction d'affectation (de la forme A=...) que la valeur sera changée.

Essayez le dialogue suivant (nous omettons les **ENTER** et les Ready) :

```
AL = 3
?AL+5
 8              (5 + 3 = 8)
?3*AL           (AL vaut toujours 3)
 9              (3 × 3 = 9)
```

Il reste encore une question : à combien de variables différentes avons-nous droit ? La limite réelle est en fait la taille mémoire, mais on peut espérer avoir plusieurs centaines de variables, c'est-à-dire de quoi traiter les problèmes les plus complexes.

ÉVALUATION DES EXPRESSIONS

On peut calculer des expressions plus compliquées renfermant plusieurs opérations. Essayez :

```
?5+4*2
?2*3↑2
?5*3/4
```

La première a pour résultat 13, c'est-à-dire qu'on a effectué d'abord la multiplication. La deuxième donne 18, car l'on a d'abord effectué le 3 ↑ 2. La troisième s'évalue en calculant d'abord 5*3 puis en divisant le résultat par 4.

En résumé, on effectue généralement de gauche à droite, mais on a une règle de priorité des opérateurs :

↑ est le plus prioritaire, puis on a :
– (prendre l'opposé, exemple : –X), puis :
* et / ex-aequo, puis :
+ et – (soustraction) ex-aequo.

Si l'on veut changer l'ordre de priorité, on emploie des parenthèse : un groupe entre parenthèses est toujours évalué en premier. Par exemple :

(2+10)/5	donne 2.4	alors que	2+10/5	donne 4
6–(3+5)	donne –2	alors que	6–3+5	donne 8

On peut avoir des parenthèses emboîtées, mais il faut toujours qu'il y ait le même nombre de parenthèses ouvrantes et fermantes et que l'expression ait un sens. Par exemple :

```
?(A–3*(B–C))/5 ↑ (B*C)
```

On dispose, pour faciliter les calculs, d'un ensemble de fonctions mathématiques qui peuvent intervenir dans les expressions arithmétiques. Leur argument est, comme toute sous-expression entre parenthèses, évalué en priorité. La liste complète de ces fonctions est donnée en annexe, mais nous citons tout de suite : SIN (sinus), COS (cosinus), SQR (racine carrée), EXP (exponentielle), etc.

```
?1+SQR(2)      donne      2.41421356
```

PRÉCISION DES NOMBRES

En examinant les résultats obtenus dans les exemples qui précèdent, nous pouvons faire un certain nombre de remarques :

– L'Amstrad peut imprimer des nombres positifs ou négatifs. Il imprime un '–' en tête pour un nombre négatif, rien pour un nombre positif.

– L'Amstrad peut imprimer des nombres entiers ou fractionnaires. Pour les nombres fractionnaires, on utilise un point (convention anglo-saxonne) au lieu de la virgule des français. On utilise le système décimal, il n'y a pas à se soucier de binaire. Signalons aussi que les zéros sont barrés (Ø) pour ne pas être confondus avec la lettre O.

– L'Amstrad imprime au maximum 9 chiffres significatifs. Il en utilise un peu plus dans la représentation interne pour faire les calculs, mais n'en sort que 9. La représentation interne est toujours fractionnaire, mais si, à la précision des calculs près, le nombre est assez proche d'un entier, il sera imprimé comme entier.

```
?14.99999999     donne     15
```

– Si le nombre à imprimer est <0,1 ou >9999999999, il est imprimé en "notation flottante", c'est-à-dire sous la forme : Sx.xxxxxxxxEstt (x et t sont des chiffres, S est le signe du nombre, s le signe de l'exposant).

```
?0.000123          donne 1.23E-04
                   ce qui se comprend comme 1,23x10^-4

?-1000000000       donne -1E+09
                   ce qui se comprend comme -1×10^9
```

Voilà terminée notre prise de contact avec l'Amstrad. Nous espérons qu'elle n'a pas été trop ardue et nous vous encourageons à essayer d'autres exemples, pour bien vous familiariser.

Nous sommes prêts maintenant à utiliser l'Amstrad en mode programmé.

CHAPITRE 2
INSTRUCTIONS FONDAMENTALES

LE PREMIER PROGRAMME

Tout en commençant par des notions très simples, nous passons maintenant à une programmation plus élaborée, en mode différé.

Tout traitement d'informations, donc tout programme, comprend trois étapes fondamentales, immuables :

- l'acquisition ou entrée des données à traiter ;
- le traitement des données c'est-à-dire le calcul des résultats ;
- la sortie des résultats.

On aura donc en BASIC trois instructions fondamentales correspondant à ces trois actions ; on les retrouve dans le programme A qui a simplement pour but de calculer la surface d'un cercle de rayon R ($S = PI \times R^2$).

PROGRAMME A-1

```
10 INPUT R
20 S = PI*R ↑ 2
30 PRINT S
```

Les programmes sont identifiés par une lettre suivie d'un numéro de version.

L'instruction 10 correspond à l'entrée au clavier du rayon R. Lorsque vous aurez tapé RUN, en exécution de cette instruction, l'ordinateur affichera sur l'écran un ? et le curseur (carré blanc). Il se mettra en attente que vous tapiez une valeur de rayon. Lorsque vous l'aurez fait, admettons que vous ayez tapé 1.5 (ce qui veut dire 1,5) sans oublier le **ENTER**, l'Amstrad affectera cette valeur à la variable et passera à l'instruction suivante.

L'instruction 20 est l'instruction du calcul proprement dit de la surface à laquelle on a donné le nom de variable S. L'expression arithmétique BASIC est une copie presque conforme (les signes d'opération sont obligatoires) de la formule mathématique qui intervient. On voit que l'Amstrad "connaît" la valeur de π (dans la variable réservé appelée PI).

L'instruction 30 demande simplement l'affichage du résultat 7.06858347 comme nous l'avons vu en mode direct.

Tout ceci est simple n'est-ce pas ? Eh bien, cela renferme 90 % de tout le BASIC, puisque tout repose sur les trois opérations fondamentales que nous avons citées.

Bien que simple, le programme que nous venons d'écrire fonctionne de façon satisfaisante : pour calculer la surface d'un cercle, on tape RUN puis, dès que l'opérateur a affiché un point d'interrogation, on tape la valeur du rayon considéré et dès que l'on a appuyé sur la touche **ENTER**, le programme présent en mémoire s'exécute.

QUELQUES PERFECTIONNEMENTS

Toutefois, le comportement de ce programme a quelques petits inconvénients auxquels nous allons remédier, ce qui va nous permettre de voir de nouvelles instructions BASIC ou de nouvelles formes de celles que nous connaissons. Nous suivrons d'ailleurs cette méthode tout au long du livre.

Objection n° 1

Lorsque l'Amstrad affiche le ? au cours de l'instruction INPUT, rien ne dit que c'est à un rayon qu'il s'attend ; cela n'est pas gênant car nous le savons, mais une personne qui ne connaîtrait pas le programme ne saurait pas quoi répondre au point d'interrogation. Même l'auteur du programme, dans le cas d'un programme qui a besoin de beaucoup de données, peut avoir besoin qu'on lui remémore dans quel ordre il faut les entrer.

En résumé, ce qui serait souhaitable, c'est d'afficher un message qui dit à l'utilisateur quelles sont les données attendues par l'INPUT. On aimerait, par exemple, avoir un affichage du genre :

Rayon du cercle ? ou Donnez un rayon ?

Eh bien, BASIC le permet très facilement. En effet, on peut employer INPUT sous la forme :

```
INPUT "Texte";Variables
```

A ce moment, le texte entre les guillemets va apparaître sur l'écran, suivi du point d'interrogation habituel de INPUT et il suffit de répondre comme dans le cas précédent.

Ainsi, si l'on remplace l'instruction 10 du programme A-1 par :

```
10 INPUT "Rayon du cercle";R
```

notre problème sera complètement résolu. Le point-virgule qui sépare le texte de la liste de variables est impératif, tout comme les guillemets qui délimitent le texte.

Oui, mais comment remplacer l'ancienne instruction 10 par la nouvelle ? Tout simplement en tapant la nouvelle instruction 10, sans oublier de commencer par le numéro. Elle viendra remplacer l'ancienne dans la mémoire où le programme est conservé.

Objection n° 2

Lorsque le résultat est affiché, on obtient un nombre, mais rien n'indique qu'il s'agisse de la surface du cercle. On aimerait bien, là aussi, qu'un message convenable nous éclaire. Dans le cas d'un programme fournissant beaucoup de résultats, il serait tout à fait indispensable que chaque résultat soit identifié.

Comme le précédent, ce problème se résout facilement en BASIC. Il suffit de savoir qu'on peut, par PRINT, afficher n'importe quel texte entouré de guillemets. Ainsi, nous pourrons remplacer l'instruction 30 par :

```
30 PRINT "Surface=" ;S
```

et tout sera dit.

On aura maintenant un dialogue de la forme (nous soulignons ce qui est tapé par l'utilisateur) :

```
RUN
RAYON DU CERCLE ? 10
SURFACE = 314.159265
Ready
```

Objection n° 3

Notre programme est nettement amélioré, surtout du point de vue du dialogue homme-machine et toute entrée/sortie doit toujours se présenter ainsi. Mais il reste encore un petit élément d'inconfort : lorsqu'on veut traiter plusieurs rayons, il faut à chaque fois taper RUN, ce qui est un peu fastidieux.

BASIC a une réponse : nous allons ajouter à la fin du programme une instruction qui dit à l'Amstrad "Recommencez l'exécution depuis l'instruction 10". Ajoutons l'instruction :

```
40 GOTO 10
```

Sachant que GOTO signifie "Aller à", le fonctionnement est évident : à chaque fois qu'un calcul est fait et son résultat imprimé, l'Amstrad arrive à l'instruction 40 qui le renvoie à 10 où l'on demande un nouveau rayon et ainsi de suite...

On dispose maintenant du programme A-2 qui, bien qu'il ait été facile à écrire, a un comportement très commode. Il fait apparaître une notion importante, la notion de boucle. Une structure formée d'un groupe d'instructions qui seront exécutées plusieurs fois s'appelle une boucle. La possibilité d'exécuter des boucles, donc d'effectuer des calculs itératifs est un point fort des ordinateurs.

PROGRAMME A-2

```
10 INPUT "Rayon du cercle";R
20 S=PI*R ↑ 2
30 PRINT "Surface = ";S
40 GOTO 10
```

SORTIE DU PROGRAMME PAR ESC
COMMANDE CONT

La boucle que nous venons de voir dans le programme A-2 résolvait un problème, mais elle nous en pose un autre : en effet, elle ne finit jamais. Indéfiniment, l'Amstrad demandera un nouveau rayon, et encore un autre...

Il est, en fait, normalement interdit d'implanter une telle boucle indéfinie dans un programme ; tout programme doit être assuré de se terminer au bout d'un temps fini. Nous verrons par la suite des instructions permettant d'établir des boucles dont on est sûr qu'elles se terminent. Cependant, on a un moyen de s'en sortir.

Ce moyen est nécessaire car, malheureusement, il peu arriver que les choses s'arrangent mal. Si, par suite d'une erreur de programmation, l'ordinateur se perd dans une boucle sans fin et tourne indéfiniment sans fournir de résultat, ne peut-on pas reprendre le contrôle ?

Si ! Il suffit d'appuyer sur la touche ESC. On obtient aussitôt l'affichage :

```
* Break *
Break in      (arrêt à l'instruction numéro...)
Ready
```

On peut alors faire imprimer des variables pour voir si tout est correct. Il se peut, en effet, que l'on n'obtienne pas de résultat pendant longtemps, simple-

ment parce ce que les calculs sont longs et non parce que l'on a une boucle indéfinie. A ce moment, on peut faire reprendre l'exécution là où elle en était, en tapant la commande en mode direct : CONT.

ESC fait arrêter l'exécution du programme de façon que celle-ci puisse reprendre par CONT. CONT ne peut fonctionner que s'il n'y a eu aucune modification du programme pendant l'arrêt.

Le programme A-2 est donc correct : nous pouvons avoir la surface de tous les cercles que nous voulons et, quand nous n'en voulons plus, nous frappons la touche ESC pour terminer.

N.B. – Il faut en fait, généralement, appuyer deux fois sur ESC. L'emploi de la touche ESC une seule fois produit un arrêt sans redonner le contrôle au clavier : on reprend par un second appui sur ESC. Lorsque l'Amstrad est en train d'exécuter un INPUT (ce qui était le cas dans notre exemple), un seul appui sur ESC suffit.

Compléments

Sans prétendre être complets (le meilleur moyen d'apprendre toutes les particularités d'un langage, c'est la pratique), il y a un ou deux détails supplémentaires que nous devons donner maintenant sur INPUT et sur PRINT.

ENTRÉE DE PLUSIEURS DONNÉES

On peut entrer plusieurs données dans une même instruction INPUT. Par exemple, si au lieu de calculer la surface du cercle, nous voulions calculer le volume d'un cylindre, il faudrait donner le rayon, mais aussi la hauteur H.

On pourrait employer une instruction de la forme :

```
10 INPUT "Rayon,Hauteur" ;R,H
```

et, en réponse, il faudrait maintenant taper deux nombres séparés par une virgule :

```
Rayon,Hauteur ? 15.5,20 ENTER
```

Ne confondez pas le point de 15.5 qui, en notation anglaise sépare partie entière et partie décimale d'un nombre, avec la virgule qui sépare deux nombres différents qui iront dans des variables différentes.

Question : en réponse à INPUT I,J où je devais entrer les valeurs 45 et 105, j'ai, par erreur, mal placé la virgule et tapé 4,5105. Que se passe-t-il ? – L'Amstrad va entreprendre les calculs avec les valeurs I=4 et J=5105 qui ne sont pas celles que vous souhaitiez.

Autre question : à INPUT I,J où je devais fournir deux valeurs, je n'en ai donné qu'une, suivie de **ENTER**. *Que se passe-t-il ?* – L'Amstrad proteste par le message "Redo from start" et vous reprenez l'entrée de vos données en essayant de ne pas en oublier, cette fois, Lorsqu'on a un INPUT à plusieurs variables, il faut fournir toutes les données voulues, puisque, tant qu'il n'aura pas eu toutes les valeurs qu'il attendait, l'Amstrad le signalera en affichant son message et vous devrez recommencer votre entrée.

Exemple de dialogue (instruction 10 INPUT "I,J" ;I,J) :

I,J ? 45 **ENTER**
?Redo from start
I,J ? 45,105 **ENTER**

Au contraire, maintenant, je tape trop de valeurs (exemple : 45,105,200 pour INPUT I,J). Que se passe-t-il ? – L'Amstrad proteste de la même manière et vous devez recommencer l'entrée des n premières valeurs s'il en attend n.

Et si je ne tape pas du tout de valeurs, c'est-à-dire si, en réponse à INPUT, je fais **ENTER** *tout de suite ?* – Là encore, l'Amstrad proteste par "Redo from start".

Dernier détail : vous pouvez remplacer le point-virgule qui sépare le texte de la liste des variables par une virgule ; à ce moment, l'Amstrad ne mettra pas de point d'interrogation.

EXPRESSION ARITHMÉTIQUE DANS UN ORDRE PRINT

Nous avons, jusqu'ici, demandé l'impression soit d'un titre, soit de la valeur d'une variable. Ce sont des cas particuliers de la loi générale qui permet de mettre comme élément à imprimer n'importe quelle expression arithmétique : l'Amstrad effectuera le calcul et affichera la valeur obtenue.

Tapez

```
?"5 A pour carré" ;5 ↑ 2
```

Vous obtiendrez

```
5 A pour carré 25
```

Comme cas particulier d'expression arithmétique, on peut mettre une variable, ou même une constante. On pourrait écrire :

```
? 5;"A pour carré" ;5 ↑ 2
```

à la place de l'exemple précédent.

Lorsqu'on fait imprimer un titre, on exploite ce fait : un texte encadré de guillemets constitue une constante chaîne de caractères.

SÉPARATION DES ÉLÉMENTS DANS UNE LISTE D'IMPRESSION

Il est bien entendu que là où nous disons "impression", il faut entendre "affichage sur l'écran", mais avec un Amstrad, il suffit d'une seule modification que nous verrons bien plus tard pour que toutes les informations concernées par tous les ordres d'impression apparaissent à l'imprimante si l'on en a une.

Finissons-en aussi avec le point d'interrogation : sur Amstrad, ? est une abréviation commode de l'instruction PRINT.

Par ailleurs, dans les différents exemples qui ont précédé, lorsque nous avons voulu afficher plusieurs informations dans le même ordre PRINT, nous les avons séparées tantôt par un point-virgule, tantôt par une virgule. Quelle est la différence ? Lorsque deux zones sont séparées par un point-virgule, elles seront imprimées côte à côte sur la ligne, tandis que si on les sépare par une virgule, l'impression de la seconde zone commencera à la colonne 14 (on appelle cela la tabulation ; sur écran, cela ne permet que trois données par ligne en mode 40 colonnes ; cette largeur est réglable par une instruction ZONE).

Essayez :

```
?A,B
 0             0
?A;B
 0  0
```

Pourquoi les zéros ne sont-ils pas "collés" dans le second exemple ? Parce que chaque nombre imprimé comporte un signe (ici, on a un espace car les nombres sont positifs) et est suivi d'un espacement.

Qu'en est-il pour les chaînes de caractères ?

Essayez :

```
?"Bon";"Jour"
Bonjour
?"Bon","Jour"
Bon             Jour
?"Eh bonjour","Monsieur"
Eh bonjour      Monsieur
```

Encore deux indications :

– Si nous ne mettions aucun séparateur, l'Amstrad ferait comme si nous avions mis un point-virgule. Mais cela n'est possible qu'entre deux chaînes de caractères ou entre chaîne et variable.

– On peut terminer l'ordre PRINT par une virgule ou un point-virgule alors qu'il n'y a rien à séparer. Cela a pour effet que le prochain ordre PRINT écrira

sur la même ligne, de façon jointive ou avec une tabulation, selon que l'on aura mis un point-virgule ou une virgule.

Essayez les programmes :

```
1              2              3
10 ?"Bon"      10 ?"Bon";     10 ?"Bon",
20 ?"Jour"     20 ?"Jour"     20 ?"Jour"
```

Effet :

```
Bon            Bonjour        Bon   Jour
Jour
```

Tout vient de ce qu'un ordre PRINT normal effectue un "retour chariot" pour terminer et celui-ci est supprimé par le point-virgule ou la virgule.

Par contre, si l'on veut avancer d'une ligne sans rien imprimer, il suffit de mettre PRINT tout court.

Précisons enfin, que si pour un PRINT, la virgule ou le point-virgule ne font pas grade différence, sauf si l'on veut soigner la mise en page, pour IMPUT, ils sont essentiels et pas du tout interchangeables : les variables à entrer doivent être séparées par des virgules (ainsi que les données au moment où on les fournit) et s'il y a un message de titre, il doit être séparé du reste par un point-virgule. Si on le sépare par une virgule, le point d'interrogation est supprimé à l'exécution.

Bien que nous n'ayons vue que trois instructions BASIC, nous avons déjà en main des possibilités immenses, car elles couvrent les trois opérations fondamentales de tout traitement :

– entrer les données,
– calculer,
– sortir les résultats.

Vérifions-le sur deux exercices que nous vous recommandons instamment de traiter sans regarder la solution à la fin du volume.

Exercice 2.1.– Modifier le programme A-2 pour calculer le volume de cylindres de rayon R et hauteur H.

Exercice 2.2– Écrire un programme de structure analogue, mais qui calcule l'intérêt rapporté par un certain capital placé à un certain taux pendant N années (intérêts composés annuellement).

Remarque : la numérotation des exercices est faite sous la forme chapitre.numéro.

CHAPITRE 3
COMMANDES FONDAMENTALES

Un programme entré en mode différé ne peut être utilisé qu'en conjonction avec un certain nombre de commandes en mode direct, qui disent au système d'exploitation ce que l'on veut faire.

Nous avons déjà vu la plus fondamentale de ces commandes : RUN qui permet d'exécuter le programme.

Mais on peut faire d'autres opérations sur un programme. On peut le lister, c'est-à-dire imprimer ses instructions, ce qui est utile, notamment pour rechercher des erreurs, ou si l'on envisage une modification. On peut le corriger et, pour cela, l'Amstrad est assez commode. On peut le sauvegarder sur cassette pour pouvoir l'utiliser ultérieurement sans avoir à le retaper.

Le BASIC de l'Amstrad possède tout un environnement de commandes permettant ces opérations. Il est indispensable que nous les voyions maintenant.

LIST

LIST, tout court, fournit sur l'écran la copie de tout le programme présent en mémoire. S'il n'y a pas de programme en mémoire, par exemple, aussitôt après la mise sous tension, l'Amstrad affiche Ok immédiatement. On dit qu'on obtient la liste, ou, en franglais, le listing du programme.

Si le programme est très long, et donc ne tient pas dans les 23 lignes de l'écran, la liste va défiler sur l'écran (une ligne apparaît en bas alors qu'une ligne disparaît en haut) et la lecture sera difficile. Pour la faciliter, on peut stopper en appuyant sur la touche ESC. Le listing reprend en appuyant sur n'importe quelle touche autre que ESC.

Vous pouvez l'essayer, mais les programmes que nous avons écrits jusqu'à présent sont un peu courts pour que cela se voie bien.

Deux particularités sont apparues sur les listes que nous avons pu obtenir en faisant quelques essais.

– Les instructions apparaissent sur la liste dans l'ordre des numéros croissants, c'est-à-dire le même ordre que pour l'exécution, quelque soit l'ordre dans lequel elles ont été tapées.

– Si, pour des instructions d'impression, nous avons utilisé le point d'interrogation, sur la liste, c'est le mot PRINT qui apparaît. Si les mots-clés ont été tapés en minuscules, il sont listés en majuscules.

Nous supposerons dans la suite de l'ouvrage que nous avons en mémoire le programme A-2 du chapitre précédent.

Listes partielles

On peut lister une seule instruction, en donnant son numéro : LIST x

```
LIST 20
20 S=PI*R ↑ 2
```

Pour lister toutes les instructions comprises entre les instructions de numéros x et y, on tape LIST x – y

```
LIST 20–30
20 S=PI*R ↑ 2
30 PRINT "Surface =" ;S
```

Les bornes - si elles existent - sont comprises.

LIST 15–35 donnerait le même résultat que l'exemple précédent.

- LIST –x : liste depuis le début jusqu'à la ligne x :

```
LIST –20
10 INPUT "Rayon du cercle";R
20 S=PI*R ↑ 2
```

- LIST x– : liste à partir de la ligne x jusqu'à la fin :

```
LIST 30–
30 PRINT "Surface =" ;S
40 GOTO 10
```

Comme l'exécution d'un programme, une liste peut être arrêtée à l'aide d'un double appui sur la touche ESC. Couplé avec LIST x–, cela permet de lister un long programme par morceaux : on appuie sur ESC quand on voit que l'écran va être plein.

Bien sûr, s'il n'y a aucune instruction dans l'intervalle demandé, on a tout de suite Ready.

NEW

Lorsqu'on tape une instruction BASIC en mode programmé (donc avec un numéro), il peut se passer deux choses :

– ou bien l'instruction que l'on vient de taper porte le même numéro qu'une instruction déjà présente : à ce moment, elle vient remplacer l'ancienne;

– ou bien il n'existait pas d'instruction de même numéro que celle qu'on vient de taper : alors - comme on pourrait le constater en demandant LIST - l'instruction vient s'intercaler dans le programme à la place indiquée par son numéro.

Il en résulte que, si l'on veut introduire un programme complètement nouveau et si les instructions nouvelles ne correspondent pas une à une à celles de l'ancien programme, il restera des instructions anciennes au milieu des nouvelles, qui, bien entendu, perturberont le fonctionnement.

La commande NEW a pour but d'éliminer cet inconvénient : son effet est de supprimer complètement le programme actuellement présent. Il est conseillé de l'utiliser avant de taper un nouveau programme.

NEW ne doit pas être confondue avec une autre commande ou instruction CLEAR qui, elle, a pour effet de remettre à zéro toutes les variables. En somme, NEW vide la mémoire programme tandis que CLEAR vide la mémoire des données (l'une et l'autre sont deux zones de la même mémoire).

Enfin, une autre instruction, CLS, vide l'écran.

Remarque : en fait, NEW contient CLEAR, c'est-à-dire que lorsqu'on fait NEW, toutes les variables sont, par la même occasion, remises à zéro. De même, RUN contient CLEAR, ce qui fait qu'au début de l'exécution d'un programme, toutes les variables ont la valeur 0 jusqu'à ce qu'une instruction leur donne une autre valeur.

RUN

Nous connaissons déjà bien la commande RUN tout court. On peut l'employer aussi sous la forme RUN x où x est un numéro d'instruction. Cela aura pour effet de lancer l'exécution du programme, mais à partir de la ligne x.

Question : j'ai un programme comportant l'instruction 10. Je peux le lancer par RUN 10. Je peux aussi le lancer en tapant GOTO 10 en mode direct. Quelle est la différence? – RUN 10 remet les variables à zéro, ce que ne fait pas GOTO 10. Entrez le programme :

```
5 A=3
10 PRINT A
```

et essayez les dialogues :

1	2	3
A=5	A=5	A=5
RUN	RUN 10	GOTO 10

Effet :

3	0	5

Dans le premier cas, on passe sur l'instruction 5 qui donne à A la valeur 3. Dans le second cas, RUN remet A à zéro. Ce n'est que dans le troisième cas que l'effet de l'affectation en mode immédiat sera conservé.

END

Le RUN numéro permet de constituer un programme en plusieurs parties telles que l'on exécute tantôt l'une tantôt l'autre. Il suffit de taper RUN numéro de la première instruction de la partie voulue.

Oui, mais supposons que l'on ait exécuté la première partie : on va tomber maintenant sur la deuxième partie, ce qui n'est peut-être pas souhaité. Il suffit de terminer chaque partie par l'instruction END qui veut dire "Retourner au niveau de commande directe". Après exécution d'une instruction END, BASIC affiche Ok. Pour la bonne règle, notre programme A-1 aurait dû se terminer par une instruction :

```
40 END
```

Mais, en fait, lorsque BASIC arrive à la dernière instruction d'un programme sans rencontrer de END, il fait comme s'il y avait cette instruction.

ÉDITION D'UN PROGRAMME

Nous devons voir maintenant tout un ensemble de procédures qui permettent de modifier ou corriger un programme en ayant le moins possible à retaper.

De ce point de vue, l'Amstrad est assez commode. Mais avant de voir ces procédures, il nous faut faire plus ample connaissance avec le clavier. Nous aurions pu le faire précédemment, avant même de procéder à nos premiers essais, mais nous avons pu nous en passer, alors que, maintenant, c'est indispensable.

Le clavier Amstrad

Un clavier Amstrad est représenté ci-après.

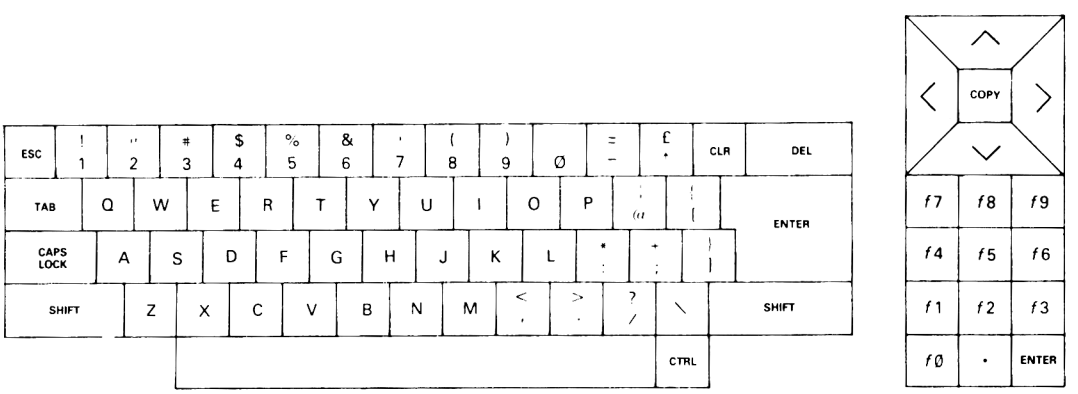


Figure 3.1. – Clavier MSX

On peut considérer qu'il y a trois sortes de touches :

– Les touches ordinaires qui font afficher le caractère correspondant sur l'écran : par exemple, quand vous appuyez sur la touche A, il s'imprime un A sur l'écran.

– Les touches de modification de l'affichage et les touches spéciales comme **ENTER** (la touche en angle) ou les touches mouvement de curseur ou encore **ESC**.

– Les touches de commande qui, enfoncées simultanément avec une seconde touche, déterminent la fonction de cette seconde touche. Les touches de commande sont **SHIFT** (il en existe deux qui sont parfaitement équivalentes), **CTRL** (Contrôle), **CAPS LOCK** (Capitales).

Chaque touche ordinaire a, en général, au moins deux fonctions représentées l'une au-dessus de l'autre, sur le dessus de la touche (*ex.* [?/]). Vous obtenez le caractère du bas (*ex.* /) en appuyant simplement sur la touche. Vous obtenez le caractère du haut (*ex.* ?) en appuyant en même temps sur **SHIFT**.

Lorsque nous disons "en appuyant simultanément", cela signifie : appuyer d'abord sur la touche de commande (*ex.* **CTRL**) et la maintenir enfoncée, appuyer sur la touche voulue, la relâcher et, enfin, relâcher la touche de commande. Entraînez-vous à votre clavier!

Touches mouvement de curseur

Le curseur est le rectangle blanc que vous avez à l'affichage lorsque l'Amstrad attend que vous tapiez quelque chose : il marque la position sur l'écran où apparaîtra le prochain caractère que vous taperez.

Les touches mouvement de curseur déplacent le curseur sans imprimer de caractère ni modifier les caractères déjà présents à l'écran, sur lesquels le curseur passe.

Les touches →, ↑, ← et ↓ créent, de façon évidente, le mouvement marqué.

Exemple

↑ que nous figurerons sous la forme h (ne pas confondre avec l'opérateur d'élévation à la puissance) fait aller d'un cran vers le haut. De même, nous utiliserons les abréviations suivantes pour les symboles flèche en bas b, flèche à gauche g, flèche à droite d.

Exercice d'entraînement 3.1. – Effectuez le parcours figure 3.2. en prenant bien soin de revenir au point A.

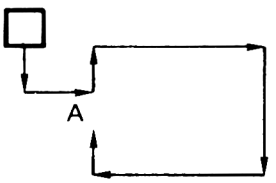


Figure 3.2.

Employées avec SHIFT, les touches curseur ont un rôle spécial que nous verrons à propos de l'édition des programmes.

La touche DEL supprime le caractère immédiatement à sa gauche, permettant ainsi une correction des erreurs de frappe.

Exemple

Nous voulons taper BONJOUR, mais nous nous apercevons que nous avons tapé BONKO.

Nous tapons une première fois sur DEL : BONK puis une deuxième : BON et nous n'avons plus qu'à continuer JOUR.

Exercice d'entraînement 3.2. – Vous vouliez taper BONJOUR et vous avez tapé BONUR.

La touche CLR joue un rôle presque identique.

Avec la touche CLR, le caractère à gauche du curseur est supprimé. Le curseur (avec le caractère qui est en dessous) et tous les caractères à sa droite, sont déplacés vers la gauche pour combler le trou.

Avec la touche CLR, c'est le caractère sous le curseur qui est supprimé et les caractères à sa droite sont déplacés vers la gauche pour combler le trou : le curseur reste donc immobile.

Ces dernières touches vont spécialement nous servir pour la correction des programmes. Il n'y a pas de touche d'insertion : nous verrons à propos de l'édition de programmes comment effectuer une insertion.

Autres touches spéciales

La barre d'espace n'est pas vraiment une touche spéciale. Elle produit le caractère ESPACE.

Nous avons déjà vu le rôle - fondamental - de la touche ENTER (retour chariot et, surtout, prise en compte de ce qu'on vient de taper). La seconde touche ENTER en bas à droite du pavé numérique, joue exactement le même rôle et elle facilite l'entrée des données numériques.

Nous avons déjà vu le rôle de ESC (pause d'un programme ou d'un listing) et de la double frappe de ESC (arrêt d'un programme et retour en mode direct).

La combinaison CTRL SHIFT ESC joue un rôle de "reset", comme si l'on éteignait et rallumait l'Amstrad : elle remet l'affichage en mode normal, ainsi que les couleurs, elle fait afficher le message de démarrage, mais surtout – faites attention – elle supprime le programme en mémoire.

La touche TAB fait imprimer une flèche à droite, ce qui n'est pas vraiment une tabulation, mais des logociels commerciaux peuvent l'utiliser comme telle.

La touche CTRL

La touche CTRL produit des fonctions spéciales lorsqu' elle est appuyée en même temps que certaines autres touches. La plupart des fonctions effectuent un formatage sur écran.

Lorsqu'on appuie simplement sur CTRL et la touche voulue, on obtient seulement l'affichage d'un caractère bizarre ; ce n'est que lorsque la combinaison de touches est incluse dans les guillemets d'un PRINT qu'on obtient la fonction spéciale. Exemple : si vous tapez CTRL G, vous obtenez le dessin d'une sonnette ; si vous tapez?"CTRL G", vous voyez sur l'écran ?"dessin de sonnette", mais lorsque vous faites ENTER, donc lorsque l'instruction est exécutée, vous entendez un bip-bip. La seule exception à ceci est CTRL M qui agit toujours tout de suite et produit un retour-chariot.

Voici les principales fonctions spéciales :

CTRL	avec I	Effet
G	I	Bip-bip
H	I	Équivalent de curseur à gauche
I	I	Équivalent de curseur à droite
J	I	Équivalent de curseur bas
K	I	Équivalent de curseur haut
L	I	Équivalent de CLS (vidage écran)
M	I	Équivalent de ENTER
S	I	Vide l'écran jusqu'au curseur
T	I	Vide l'écran à partir du curseur
X	I	Passage en contraste inversé. Un deuxième CTRL X ramène au contraste normal
↑	I	Ramène le curseur en haut à gauche de l'écran

Avec CTRL, la touche curseur gauche ramène au début de la ligne et la touche curseur droite amène à la fin de la partie écrite de la ligne.

La touche CAPS LOCK

Une première particularité est qu'il s'agit d'une touche à verrouillage. On quitte l'état "capitales" en appuyant à nouveau sur la touche CAPS LOCK.

L'effet de la touche CAPS LOCK est très semblable à SHIFT mais elle n'agit que sur les lettres. En mode normal, une touche lettre donne la minuscule; avec SHIFT, on obtient la majuscule. En mode CAPS, on obtient la majuscule, qu'il y ait SHIFT ou non. Mais pour une touche qui n'est pas une touche lettre, CAPS est inopérante. Par exemple, avec la touche [<,], même en mode CAPS pour obtenir <, il faut SHIFT.

Majuscules et minuscules

Clarifions une fois pour toutes le problème des majuscules et minuscules. Pour les mots-clés BASIC, majuscules et minuscules sont équivalentes. Vous pouvez taper les mots-clés en minuscules, en majuscules (et là, que ce soit avec **SHIFT** ou avec **CAPS LOCK**, c'est pareil) ou même mélanger les deux, au listing les mots-clés apparaîtront en majuscules. C'est pourquoi, dans ce livre, nous les écrivons en majuscules.

Au contraire, dans une chaîne de caractères entre guillemets, les majuscules et les minuscules restent distinguées.

Exemple

Si vous tapez

```
10  PrInT   "Bonjour"
        ↑       ↑
   majuscules   majuscule
   et minuscules
   mélangées
```

LIST vous donne

```
10  PRINT   "Bonjour"
    tout majuscule  restées
                    minuscules
```

et RUN produit

```
Bonjour
↑ minuscules
majuscule
```

Pour les noms de variables, le BASIC de l'Amstrad a une conduite mitigée : d'une part, la majuscule et la miniscule correspondante sont parfaitement équivalentes, mais, au listing, BASIC conservera l'orthographe que vous aviez adoptée lors de la frappe. Ainsi, A ou a, c'est la même variable ; mais si vous avez tapé a, BASIC listera a et il listera A si vous avez tapé A. Maintenant, vous pouvez mélanger les deux écritures, BASIC reconnaîtra bien qu'il s'agit de la même variable. Exemple : le programme

```
10 A=5
20 print a
```

sera listé

```
10 A=5
20 PRINT a
```

et il imprimera bien 5 comme résultat.

Répétition

Toutes les touches ont la répétition automatique, c'est-à-dire que si on les maintient appuyées, leur fonction se répète. C'est spécialement pratique pour faire voyager rapidement le curseur.

Les touches de fonction

Les touches du pavé numérique sont redéfinissables par l'utilisateur à l'aide de l'instruction KEY :

```
KEY 128+n,"chaîne"
```

donne à la touche chiffre n du pavé numérique la valeur "chaîne". Ainsi, si vous faites :

```
KEY 128+5,"list"
```

il vous suffira ensuite de taper sur le 5 du pavé numérique pour obtenir list. (Le 5 du haut du clavier a, lui, garde la valeur "chiffre 5", heureusement !)

Vous pouvez même inclure le **ENTER** en faisant :

```
KEY 128+5,"list"+CHR$(13)
```

De cette manière, la combinaison **CTRL ENTER** du pavé numérique a reçu par défaut la valeur RUN " **ENTER**.

Il faut noter, sur l'Amstrad, on peut changer la valeur de n'importe quelle touche et lui donner la valeur d'une autre par KEY DEF. Ceci est utile pour créer un clavier AZERTY accentué, mais cela sort du cadre de ce livre.

Il faut noter aussi que l'Amstrad est capable d'afficher tout un ensemble de caractères graphiques qui ne s'obtiennent pas au clavier. Comme nous le verrons aux chapitres 5 et 6, ils s'obtiennent à l'aide de CHR$.

Exemple

Essayez

```
?CHR$(&hf9)
```

vous affichez un petit bonhomme.

Modification ou correction d'un programme

Lorsqu'on a trouvé une erreur dans un programme, ou lorsqu'on veut simplement y apporter une modification, il est important de pouvoir le faire commodément, c'est-à-dire en ayant le moins possible d'informations à retaper.

Nous savons déjà :

– Ajouter ou insérer une nouvelle instruction : il suffit de taper l'instruction en lui attribuant un numéro compris entre ceux des instructions entre lesquelles on veut l'intercaler.

Il en résulte immédiatement un conseil : lorsqu'on écrit la première version d'un programme, il ne faut pas donner des numéros consécutifs, afin de pouvoir faire des insertions par la suite. On suggère, par exemple, de numéroter de 10 en 10.

– Transformer complètement une instruction : on tape la nouvelle version avec le même numéro que l'ancienne. Dès qu'on tape le **ENTER**, le remplacement s'effectue.

Voyons maintenant de nouvelles possibilités :

– Suppression complète d'une instruction : il suffit de taper le numéro suivi de **ENTER**.

Notons que ceci est un cas particulier de la procédure précédente : on crée une nouvelle version vide de l'instruction et BASIC ne garde pas une instruction vide.

Pour la suite nous supposons que nous avons le programme A-1 en mémoire (sinon, retapez-le : c'est fastidieux, mais nous verrons bientôt comment récupérer un programme sans le retaper, grâce aux cassettes).

Faisons un LIST, mais, avant, nous tapons CLS pour avoir le listing en haut de l'écran.

Ensuite, nous tapons 30 **ENTER** pour supprimer la dernière instruction. Nous vérifions par LIST que cela a bien été fait. Nous n'avons plus l'instruction 30.

Pour la récupérer, si par suite d'une erreur ce n'était pas celle-là qu'il fallait supprimer, on doit théoriquement la retaper. Eh bien non! Pas tant que l'instruction est affichée. En effet, tant que l'instruction est affichée, elle est dans la mémoire d'écran.

Il y a un moyen de la transférer dans la mémoire programme.

On utilise la technique du "curseur de copie". Pour cela, vous remontez sur la ligne 30 encore en haut de l'écran, mais attention, vous faites le mouvement par **SHIFT** 'curseur haut'. A ce moment, vous voyez que le curseur se subdivise : il reste un curseur là où il était (c'est le curseur normal), tandis qu'un second curseur (c'est le curseur de copie) remonte. Lorsque ce dernier est arrivé sur le 3 de la ligne 30, vous appuyez sur la touche **COPY**. A chaque appui sur la touche **COPY**, un caractère de la ligne 30 est recopié à l'emplacement du curseur

normal. Lorsque toute la ligne est recopiée, faites **ENTER** : le curseur de copie disparaît et vous pouvez vérifier par LIST que la ligne 30 et revenue dans le programme.

– Modification de quelques caractères sur une ligne : c'est là que se manifeste toute la puissance du système d'édition de l'Amstrad. Il y a deux procédures possibles : l'utilisation du curseur de copie et la commande EDIT.

Avec le curseur de copie, la marche à suivre est la suivante :

1.– Lister la ligne à modifier (si elle n'est pas déjà sur l'écran);

2.– Amener le curseur de copie (donc **SHIFT** touche mouvement) sur la ligne à modifier et aller par **COPY** jusqu'au premier caractère à changer;

3.– Pour chaque caractère à conserver, le copier par **COPY**. Pour chaque caractère à supprimer, le passer par **SHIFT** curseur droite. S'il y a des caractères à insérer, tapez-les simplement : vous les verrez apparaître sous le curseur normal. D'ailleurs, la ligne du curseur normal vous montre l'aspect de la ligne corrigée.

4.– Lorsque la ligne a l'aspect voulu, terminer par **ENTER** après avoir copié la fin de la ligne si elle était à garder inchangée.

On peut modifier une ligne en autant d'étapes qu'on veut : il suffit de revenir sur la ligne et d'y faire d'autres modifications après avoir tapé **ENTER**. Il est conseillé de vérifier par LIST que les modifications souhaitées ont bien été faites : en particulier, il est fréquent d'oublier de terminer par **ENTER** et, à ce moment, les modifications ne sont pas enregistrées.

Avec la commande EDIT, la procédure est la suivante :

1.– Tapez EDIT numéro de ligne **ENTER**. La ligne spécifiée est listée et vous avez le curseur (normal) sur son premier caractère.

2.– Pour chaque caractère à conserver, passez-le par curseur droite. Pour chaque caractère à supprimer, amenez le curseur sur lui et faites **CLR**. S'il y a des insertions à effectuer, tapez les caractères voulus : l'insertion se fait immédiatement à gauche du curseur.

3.– Lorsque la ligne a l'aspect voulu, terminer par **ENTER**. Il n'est pas nécessaire d'aller jusqu'au bout de la ligne.

– Modification du numéro d'une ligne : c'est un cas particulier du précédent ; c'est maintenant le numéro qu'on change, mais attention! Une fois qu'on a tapé **ENTER**, il y a deux lignes identiques; l'une a l'ancien numéro et l'autre a le nouveau. Cela peut être spécialement précieux pour économiser du temps de frappe lorsque l'on a toute une série de lignes très voisines à taper.

Supposons qu'on veuille avoir (c'est un cas d'école) :

```
50 GOTO 10
70 GOTO 10
90 GOTO 10
95 GOTO 12
```

On tapera :

50 GOTO 10 ENTER

puis, curseur de copie sur le 5 :

7, SHIFT curseur droite, des COPY et ENTER

puis, curseur de copie sur le 7 :

9, SHIFT curseur droite, des COPY et ENTER

puis, curseur de copie sur le 9 :

COPY.5, SHIFT curseur droite, des COPY jusqu'au 0, 2 et ENTER..

A chaque fois, on n'aura sur l'écran que la dernière ligne tapée, mais en faisant LIST, on s'aperçoit que toutes les lignes voulues sont bien présentes.

Exercice 3.3. – Modifiez le programme A-1 pour qu'il calcule, non pas la surface d'un cercle, mais le volume de la sphère de rayon R.

Nous donnons l'exercice moins pour le calcul que pour s'entraîner à effectuer la modification. On voit facilement qu'il suffit de changer le nom du résultat en V (il serait possible de garder S, mais nous voulons des identificateurs parlants). Donc, 30 doit devenir 30 PRINT V tandis que 20 doit devenir 20 V=4/3*PI*R ↑ 3.

La modification de 30 est évidente, on amène le curseur de copie sur le S par COPY et on tape V ENTER.

Pour l'instruction 20, procédons par EDIT 20 pour changer. On amène le curseur sur le S qu'on change en V, en tapant CLR V puis, curseur sur le P, taper 4/3*. On amène ensuite le curseur jusqu'au 2 sur lequel on tape CLR et 3.

Question : ne manque-t-il pas quelque chose? – Si! Le ENTER.

Bien sûr, il aurait mieux valu modifier le programme A-2; c'est l'objet de l'exercice 3.4.

Exercice 3.4. – Faire la même modification sur le programme A-2.

Tout devrait bien se passer.

Exercice d'entraînement 3.5. – Faire le passage du programme A-2 au calcul du volume du cylindre.

Avant de voir les commandes qui concernent les cassettes, nous passons à un autre exercice qui va nous permettre d'aborder une autre sorte de problèmes.

Exercice 3.6. – Écrire un programme analogue au programme A-2, mais qui, cette fois, demande la surface d'un cercle et en déduire le rayon.

Le problème est que, cette fois, nous n'avons pas une formule connue "toute cuite" à appliquer. Il faut la chercher, encore que, pour cet exercice, ce ne soit pas trop ardu.

Néanmoins, et ce sera vrai dans tous les problèmes autres que ceux qui sont totalement évidents, l'ordinateur n'est pas capable de trouver tout seul la solution d'un problème. Il faut la lui donner sous forme d'une suite ordonnée d'opérations à effectuer. Une telle suite, qui n'est rien d'autre qu'une recette, s'appelle chez les informaticiens savants un algorithme.

Il est presque toujours plus difficile de trouver l'algorithme résolvant un problème que de programmer cet algorithme une fois qu'on l'a trouvé.

Revenons à notre exercice. On trouve facilement que la formule à appliquer est R=√S/PI. Comment allons-nous traduire la racine carrée? Eh bien, nous avons le choix, ce qui arrive souvent en programmation, entre ↑ .5 (puissance 1/2), ou appeler la fonction mathématique SQR (racine carrée) qui est une de celles dont on dispose en BASIC. D'où deux solutions pour l'instruction 20 du programme A-3.

PROGRAMME A-3

```
10 INPUT "Surface du cercle";S
20 R=SQR(S/PI)
30 PRINT "Rayon =" ;R
40 GOTO 10
```

Autre forme de 20 :

```
20 R=(S/PI)↑.5
```

RANGEMENT D'UN PROGRAMME SUR CASSETTE

Sur un 664, les commandes que nous décrivons ici agiront de la même manière mais sur disquette. La seule différence est que vous n'aurez pas les messages vous demandant de manœuvrer les touches du magnétophone. Si vous disposez en plus d'un magnétophone et avez fait le branchement décrit dans la notice, tapez la commande |TAPE pour que les sauvegardes ou chargements agissent sur cassette.

Voici maintenant des commandes spécialement utiles. En effet, jusqu'ici nous n'avons écrit que des programmes très courts et peu nombreux. Néanmoins, même ainsi, il est fastidieux de les taper plusieurs fois et cela entraîne des risques d'erreurs. Or, si nous faisons NEW, ou si nous éteignons l'Amstrad simplement pour aller nous coucher, le programme est perdu (la mémoire vive RAM perd ses informations lorsqu'elle n'est plus alimentée). Heureusement, il y a les cassettes sur lesquelles on peut sauver des programmes et les relire par la suite.

Sauvegarde

Munissez-vous d'une cassette vierge rembobinée (sinon, vous la rembobinez avec la touche REW du magnétophone). Notons à ce propos, qu'il est recommandé d'utiliser des cassettes courtes (il vaut mieux, étant donné les temps de lecture, ne ranger que peu de programmes sur chaque cassette) et de bonne qualité.

Supposons que nous ayons un programme précieux en mémoire, par exemple le programme A-3 que nous venons de faire. Placez la cassette dans le magnétophone et tapez :

```
SAVE "Rayon"
```

L'Amstrad vous dit alors :

```
Press REC and PLAY then any key
```

appuyez sur les touches REC et PLAY du magnétophone (simultanément) puis, lorsque c'est fait, sur n'importe quelle touche du clavier (en pratique **ENTER**) ou sur **ESPACE**. Le magnétophone démarre et l'Amstrad affiche :

```
Saving RAYON block 1
```

(sauvegarde ... bloc n° 1 – si le programme était long, il y aurait d'autres numéros de blocs).

Lorsque Ready et le curseur réapparaissent, c'est fini : le programme a été écrit sur la cassette sous le nom que nous avons donné. (à part qu'il est mis en majuscules).

Remarque : le nom peut comporter le nombre de caractères que vous voulez, mais sur disque, vous êtes soumis aux règles du système disque (au maximum 8 caractères + éventuellement . et 3 caractères d'extension).

Les deux vitesses de sauvegarde

La commande SPEED WRITE 1 fait effectuer les sauvegardes deux fois plus vite. La commande SPEED WRITE 0 ramène à la vitesse par défaut qui est plus lente mais aussi plus fiable.

Vérification

Une fois la sauvegarde effectuée, il est bon de vérifier qu'elle s'est bien passée tant que le programme est encore en mémoire. Pour cela, rembobinez votre cassette, mettez le magnétophone en lecture et tapez :

CAT

L'Amstrad répond

Press PLAY then any key

(appuyez sur PLAY puis sur n'importe quelle touche). Lorsque vous avez obéi, le magnéthophone démarre et l'Amstrad affiche une ligne pour chaque programme trouvé, avec le nom, les numéros de blocs, le signe $ (il y a d'autres signes possibles si le programme est sauvé avec d'autres options que nous n'étudions pas ici) et enfin la mention Ok qui signifie que le programme est bien lisible. Vous devez terminer en faisant ESC pour arrêter le déroulement de la bande (l'Amstrad n'arrête pas automatiquement).

S'il n'y a pas Ok pour votre programme, il faut recommencer la sauvegarde sur une autre cassette. Si l'ennui persiste, il faut faire vérifier votre magnétophone ou votre Amstrad.

Chargement

Pour renvoyer en mémoire un programme préalablement sauvegardé sur cassette, on utilise la commande LOAD "nom" donc ici LOAD "RAYON"..

L'Amstrad répond comme pour CAT puis lorsque vous avez obéi :

Found	nom	si RAYON n'est pas le premier programme sur la cassette, puis
Loading	RAYON	et, espérons-le,
Ready.		

On peut alors lister le programme ou l'exécuter par RUN.

Si l'on a un diagnostic d'erreur, il faut rembobiner et réessayer, puis refaire l'essai avec une autre cassette sur laquelle on aura pris la précaution de faire une seconde sauvegarde.

Noms abrégés

On peut ne pas spécifier de nom dans la commande LOAD. Par exemple, on aurait pu écrire :

LOAD"

L'Amstrad chargera le premier programme rencontré sur la bande.

On eput faire de même avec SAVE, ce qui crée un fichier sans nom, mais ce n'est pas recommandé. Sur disque, il est obligatoire de spécifier un nom. Pour LOAD, vous pouvez ne donner que les premiers caractères du nom, suivis d'une * : on chargera le premier programme de nom compatible trouvé sur la disquette.

Chargement et exécution réunis

Un programme peut être lu par RUN "nom" et alors son exécution démarre automatiquement. RUN " ou CTRL petite touche ENTER charge et démarre le premier programme trouvé sur cassette ou disquette.

Récapitulation

Nous avons maintenant vu les instructions les plus fondamentales des programmes :

arithmétique
INPUT - PRINT
GOTO - END

Nous avons vu les commandes les plus utiles :

RUN et LIST
SAVE et LOAD

ainsi que les procédures de correction des programmes.

Si l'énoncé de certains des mots-clés précédents n'éveille aucun écho en vous, nous vous conseillons, avant de poursuivre, de relire les pages qui les concernent.

Nous sommes maintenant prêts à aborder la seconde série de programmes qui va nous permettre - toujours par variantes successives - d'augmenter notre "arsenal" d'instructions BASIC.

CHAPITRE 4
BASES DE LA PROGRAMMATION

Si vous le voulez bien, nous allons jouer à un jeu. Les programmes de jeu forment une classe importante parmi les programmes de micro-ordinateurs. Tous ne sont pas débiles : certains sont très élaborés et souvent très amusants. D'autre part, la présentation sous forme de jeu de certains programmes pédagogiques les rend plus attrayants, donc plus efficaces. Nous souhaitons qu'à la fin de la lecture de ce livre, vous soyez capables, vous aussi, d'en écrire.

Pour le moment, notre jeu sera au départ un peu simple, mais il s'améliorera progressivement. Il s'agit du jeu "Devinez un nombre". Le programme connaît un nombre (fixe) et il lit la devinette du joueur; si le joueur a bien deviné, il affiche "Gagné", sinon il affiche "Perdu".

L'INSTRUCTION IF

Pour réaliser ce qui est demandé, de quoi avons-nous besoin? D'une instruction capable de tester une condition (qui sera ici nombre donné par le joueur = nombre caché), c'est-à-dire apprécier si elle est vraie ou fausse. Si elle est vraie, nous irons à une certaine partie du programme (ici, afficher "Gagné"), sinon nous irons à un autre endroit du programme (ici, afficher "Perdu").

Cette instruction existe, c'est l'instruction IF. Sa forme principale est :

```
n IF condition THEN instruction
n'...
```

Le comportement est le suivant : si la condition est vraie, on effectue l'instruction qui suit THEN, puis l'instruction de la ligne suivante n'; si la condition est fausse, on passe directement à l'instruction de la ligne n'.

On rencontre un cas particulier lorsque l'instruction qui suit THEN est un GOTO :

```
n IF condition THEN GOTO n''
n'
...
n''
```

A ce moment-là, si la condition est vraie, on va en n''; si la condition est fausse, on va en n'.

En fait, il suffit d'écrire l'un des deux mots THEN ou GOTO.

Exemple :

```
10 IF A = B THEN 50
```

ou

```
20 IF A+B<C GOTO 100
```

Nous sommes maintenant prêts à écrire notre première version du programme "Devinez un nombre".

PROGRAMME B-1

```
20 INPUT "Devinez un nombre" ; A
30 IF A = 3.25 GOTO 60
40 PRINT "Perdu !"
50 END
60 PRINT "Gagné"
```

Les instructions INPUT et PRINT sont déjà familières. L'instruction END fait terminer le programme une fois qu'on a inscrit "Perdu !". Il n'y en a pas besoin après 60, puisque c'est la dernière instruction du programme.

L'instruction IF que nous employons est du troisième type : IF...GOTO. Le nombre à deviner est 3.25 ; le nombre proposé, lu au clavier, est A ; la condition à tester est "Est-ce que A est égal à 3.25 ?" Eh bien, cela s'écrit A=3.25. C'est simple.

Du point de vue du fonctionnement du jeu, il y a beaucoup d'objections à formuler sur le programme B-1. Nous le ferons bientôt et cela nous aidera à découvrir de nouvelles instructions BASIC. Mais la forme présente nous a permis d'utiliser l'instruction IF qui est l'une des plus importantes de BASIC et nous avons encore quelques éléments à voir sur cette instruction, en particulier les différentes conditions qui peuvent suivre le IF.

La première forme de condition est : expression arithmétique relation expression arithmétique, comme 2*A+4<B ↑ 3.

Chacune des expressions arithmétiques est évaluée avant l'examen de la relation. Les opérateurs de relation utilisables sont :

=	égal	<>	différent
<	inférieur	<=	inférieur ou égal
>	supérieur	>=	supérieur ou égal

"Différent" s'écrit "Inférieur ou supérieur", ce qui ne manque pas de logique.

La deuxième forme de condition est une combinaison de relations de la première forme à l'aide des opérateurs logiques AND (et), OR (ou) et NOT (non).

c1 AND c2	est vraie seulement si les conditions c1 et c2 sont toutes les deux vraies;
c1 OR c2	est vraie dès que l'une au moins des conditions c1 ou c2 est vraie;
NOT c	est vraie si c est fausse et vice versa.

– Aller en 100 si à la fois C est supérieur à 2*A+4 et B est inférieur à 3 :

```
IF C>2*A+4 AND B<3 GOTO 100
```

– Imprimer OUI si X est extérieur à l'intervalle [1,2[(c'est-à-dire X<1 ou X>=2) :

```
IF X<1 OR X>=2 THEN PRINT "OUI"
```

Le BASIC Amstrad a aussi l'opérateur logique XOR (ou exclusif) : C1 XOR C2 est vraie si C1 ou C2 est vraie mais pas les deux.

Exercice 4.1. – Reprenez le programme A-3. Essayez de fournir une surface négative.

On obtient le message :

```
Improper argument in 20
```

ce qui est normal, puisque l'on cherche à calculer la racine carrée d'un nombre négatif : il n'y a pas de cercle de surface négative.

Un programme bien écrit doit se garantir contre de telles erreurs de l'opérateur : par exemple, un programme de jeu d'échecs doit vérifier que le coup proposé par le joueur est légal.

L'objet de l'exercice est d'installer une telle garantie dans le programme A-3 : ajoutez au programme une instruction qui renvoie en 10 demander une autre surface tant que la surface fournie n'est pas positive. On peut également, en outre, imprimer un message de protestation.

IF... THEN... ELSE

Lorsque le test a la forme d'une alternative (si condition, alors faire ceci, sinon faire cela), le MSX permet d'éviter les GOTO que donnerait la traduction :

```
      IF condition GOTO n
      cela : GOTO p
n     ceci
p     suite
```

On peut en effet écrire :

```
IF condition THEN ceci ELSE cela
```

"ceci" et "cela" doivent être des instructions BASIC (ou *cf.* page 73, des suites d'instructions séparées par des "deux-points").

Exemple : IF A<B THEN PRINT "A<B" ELSE PRINT "A>B"

Le seul impératif est que tout l'ensemble forme une seule ligne BASIC (au maximum 255 caractères). Après l'exécution de la ligne, on se retrouve à la ligne suivante après avoir fait l'une des deux branches de l'alternative. Ainsi, la séquence :

```
10 IF A<B THEN PRINT "A" ELSE PRINT "B"
20 PRINT "C"
```

fait imprimer soit	A	soit	B
	C		C

Ce n'est que si l'une des branches contient un GOTO qu'on ne se retrouve pas à la ligne suivante.

Exemple

Calculer Z = valeur absolue de X de trois façons différentes :

1	2	3
50 Z=ABS(X)	50 Z=X	50 IF X<0 THEN Z=−X
	60 IF X<0 THEN Z=−X	ELSE Z=X

Bien sûr, notre programme B-1 pourrait s'écrire :

Programme B-1B

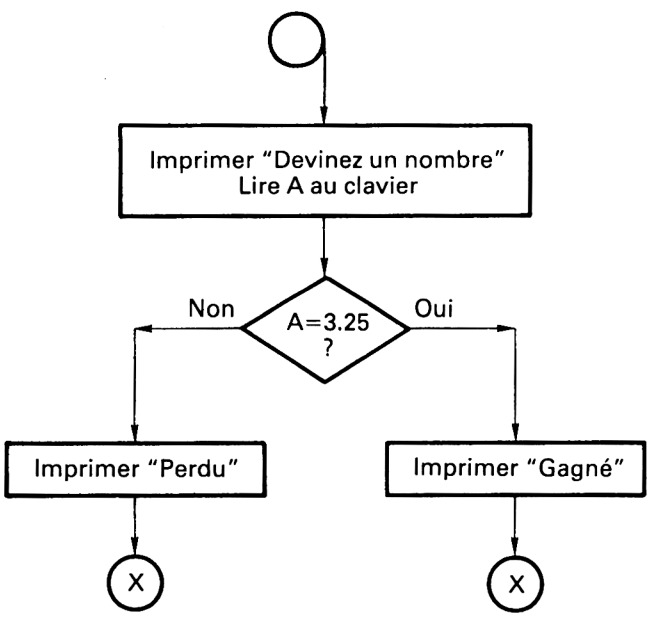
```
20 INPUT "Devinez un nombre" ;A
30 IF A=3.25 THEN PRINT "Gagné" ELSE PRINT "Perdu"
```

Nous venons maintenant de voir notre première instruction véritablement élaborée. En effet, elle rend l'ordinateur capable de prendre des décisions en fonction des différentes situations qui peuvent résulter de données. En fait, c'est vous qui prenez les décisions en préparant le programme et si jamais vous oubliez un cas possible, le programme se comportera incorrectement si ce cas se trouve réalisé.

De décision en décision, le cheminement peut se ramifier de façon complexe.

Les ordinogrammes, encore appelés organigrammes, peuvent alors être nécessaires pour s'y retrouver.

Avant de passer aux améliorations de notre programme de jeu, nous allons étudier l'ordinogramme du programme B-1.



Il est formé de blocs, qui spécifient les différentes opérations, reliés entre eux par des flèches qui représentent l'ordre de succession des opérations. La forme même du bloc indique au premier coup d'œil la nature de l'opération.

Parmi les formes de bloc, on distingue essentiellement :

– le rectangle, qui a une seule entrée et une seule sortie et représente toute opération impérative;

– le losange, qui a une seule entrée, mais deux ou plusieurs sorties, et représente les opérations de test.

On utilise les signes spéciaux  et  pour marquer le début et la fin du traitement.

Il est toujours recommandé de tracer l'ordinogramme avant d'entreprendre la rédaction du programme : ce n'est jamais une perte de temps.

Exercice 4.2. – Tracez l'ordinogramme du programme A-2. Tracez l'ordinogramme de l'exercice 4.1.

Exercice 4.3. – A quels mots-clés BASIC vus jusqu'à présent correspondent les blocs ou signes :

PERFECTIONNEMENTS DU PROGRAMME B

Il faut bien avouer que, dans sa première version, le jeu n'est pas particulièrement intéressant. Mais nous sommes maintenant en mesure de l'améliorer.

La première objection ne sera résolue que tout à fait à la fin. C'est dommage car elle concerne la crédibilité même du jeu.

En effet, il suffit de taper LIST pour avoir le listing du programme, et par là-même prendre connaissance du nombre à deviner. Nous supposerons un certain temps que le joueur ne connaît pas la commande LIST, ou alors qu'il "joue le jeu".

Il faut noter que cette objection se présente aussi, même dans des programmes plus élaborés : par exemple, si vous jouez à la bataille navale contre l'ordinateur et si vous connaissez bien le programme, vous pouvez faire imprimer les variables qui contiennent les coordonnées des navires adverses.

En supposant maintenant que vous jouez sans tricher, vous allez objecter que le jeu est très difficile, voire non équitable : vous avez peu de chances de trouver le nombre du premier coup, ce qu'exige le programme. Il est clair qu'il faut laisser plusieurs chances au joueur.

Cela peut se faire très simplement : il suffit de remplacer l'instruction 50 du programme B-1 par 50 GOTO 10 et l'on a maintenant droit à un nombre illimité de tentatives.

Malgré cela, le jeu reste bien hasardeux pour deux raisons :

– en cas d'échec, rien ne dit au joueur s'il est loin ou près du résultat;

– le joueur doit tomber pile sur la bonne valeur, c'est-à-dire doit fournir le nombre à la précision près à laquelle l'Amstrad travaille, soit 10^{-9} près.

Nous résoudrons le premier problème en calculant et en imprimant à chaque fois le pourcentage d'erreur E égal au nombre proposé moins le nombre à trouver, divisé par le nombre à trouver, le tout multiplié par cent.

Le pourcentage d'erreur sera imprimé en valeur absolue, donc sans indiquer le sens de l'erreur, pour laisser une certaine difficulté au jeu.

En ce qui concerne l'erreur possible due au manque de précision des micro-ordinateurs, on ne comparera pas le nombre proposé au nombre à trouver, mais on considèrera que la réponse est exacte si E, pourcentage d'erreur, est inférieur à 0,5 %.

Le problème de précision se pose à chaque fois que l'on a des calculs à effectuer. Pour le résoudre, il suffit de remplacer un test d'égalité pure du type IF A=B par un test sur la valeur de la différence entre les deux nombres, du type $A-B < S$ où le seuil S est l'ordre de grandeur de la précision - ou plutôt de l'imprécision ! - du micro-ordinateur. C'est d'ailleurs en fait la valeur absolue de la différence qu'il faudrait tester.

Notre programme devient :

```
1 REM  Programme B-2
2 REM
20 INPUT "Devinez un nombre";A
25 E=100*ABS(A-3.25)/3.25
30 IF E<0.5 GOTO 60
40 PRINT "Erreur";E;"%"
50 GOTO 20
60 PRINT "Gagne'!"
```

A la ligne 25, nous utilisons la fonction ABS (valeur absolue) qui fait partie de la bibliothèque mathématique du BASIC Amstrad dont la liste complète est donnée en annexe.

Si nous faisons tourner le programme tel qu'il est, nous obtenons, par exemple, l'affichage :

Erreur 10.7692308 %

Il est bien évident que nous n'avons que faire de tant de décimales. Comment les supprimer? Nous utiliserons une autre fonction de la bibliothèque, la fonction INT qui prend la partie entière de l'argument cité entre parenthèses.

Dans l'instruction 40, remplaçons E par INT(E). Cette fois, nous obtenons un pourcentage entier. Là, c'est trop peu. Comment faire pour garder - disons - deux décimales?

Pour cela, nous multiplions E par 100 pour faire passer les deux décimales que nous voulons garder dans la partie entière, nous prenons le INT du produit, ce qui fait disparaître les autres décimales, puis nous redivisons par 100 :

INT(E*100)/100

Telle est l'expression qui vient remplacer E dans l'instruction 40 du programme B-2 et nous avons maintenant un affichage satisfaisant.

Exercice 4.4. – On veut imprimer le nombre X avec D décimales. Écrire l'instruction correspondante en mode direct.

Pour constituer la version 3 de notre programme, nous voulons ajouter encore un petit perfectionnement. Il serait agréable, lorsque le nombre à deviner est trouvé, d'imprimer le nombre de tentatives qui ont été nécessaires. C'est en somme le score du jeu. Pour cela, nous introduisons une nouvelle variable N, à laquelle nous ajoutons 1 à chaque fois qu'une tentative est faite sans succès et que nous imprimons à la fin; d'où le programme B-3 :

```
1 REM  Programme B-3
2 REM
20 INPUT "Devinez un nombre";A
25 E=100*ABS(A-3.25)/3.25
30 IF E<0.5 GOTO 60
35 N=N+1
40 PRINT "Erreur";INT(E*100)/100;"%"
50 GOTO 20
60 PRINT "Gagne' en";N+1;"coups"
```

L'instruction 35 peut sembler paradoxale, mais n'oublions pas que le signe = n'a pas en BASIC le même sens qu'en mathématiques. En BASIC, il signifie : calculer l'expression qui est à droite, donc N+1 et mettez le résultat dans la variable qui est à gauche. Ici, c'est la même variable, ce qui est parfaitement licite et cela revient bien à incrémenter N.

Question : la toute première fois qu'on passe sur l'instruction 35, on doit calculer l'expression N+1; quelle valeur prend-on pour N? – Vous avez raison de poser cette question qui soulève le problème de l'initialisation des variables. Mais nous savons que lorsque l'on fait RUN, BASIC met toutes les variables - dont N - à 0. Or, lorsque l'on commence, on a fait 0 tentative, donc la valeur initiale automatique de N convient. Si, dans un autre problème, l'on avait eu besoin d'une autre valeur initiale que 0, alors il aurait fallu la fournir explicitement dans les premières instructions du programme; ce point est fondamental : beaucoup de programmes échouent pour initialisation incorrecte de certaines variables.

On remarque enfin, en 60, que l'on imprime N+1 et non N : c'est pour comptabiliser la dernière tentative, car lorsque le nombre est bon,on ne passe pas sur l'instruction 35.

Exercice 4.5. – Comment faire imprimer quand même N en 60?

LES BOUCLES FOR... NEXT

Nous pouvons maintenant nous attaquer à un autre défaut du programme dans son état actuel : il permet un nombre illimité de tentatives. C'est trop, bien sûr. Il faut autoriser plusieurs tentatives, mais en nombre limité, par exemple 5 ou 10.

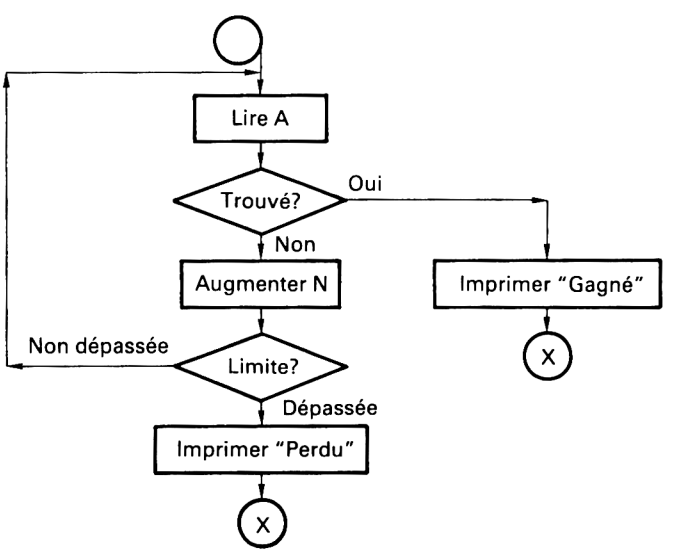
Cela peut se faire très simplement. En effet, nous avons, à tout moment, une mesure du nombre de tentatives faites jusque-là : c'est la variable N; il suffit de la tester. On remplacera, pour ce faire, l'instruction 50 par la séquence :

```
45 IF N<10 GOTO 20
50 PRINT "Je regrette. Vous avez perdu"
55 END
```

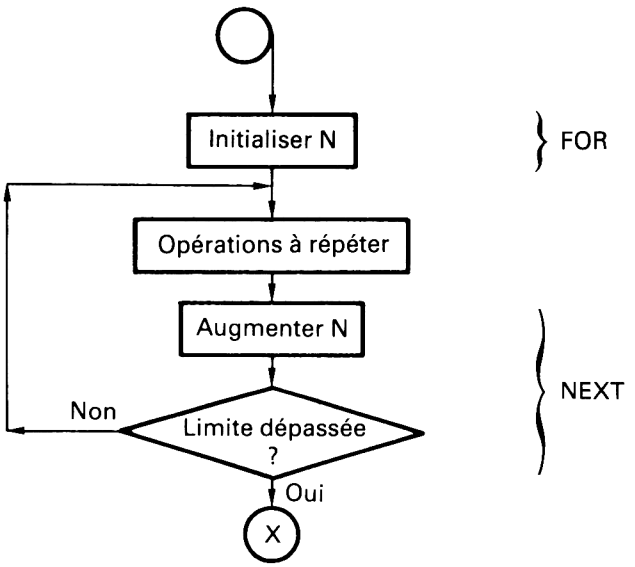
d'où le programme B-4A :

```
1 REM  Programme B-4A
2 REM
20 INPUT "Devinez un nombre";A
25 E=100*ABS(A-3.25)/3.25
30 IF E<0.5 GOTO 60
35 N=N+1
40 PRINT "Erreur";INT(E*100)/100;"%"
45 IF N<10 GOTO 20
50 PRINT "Je regrette. Vous avez perdu."
55 END
60 PRINT "Gagne' en";N+1;"coups"
```

Dessinons l'ordinogramme correspondant sans trop le détailler :



Si nous simplifions encore cet ordinogramme pour en faire ressortir l'ossature, nous obtenons :



Cette structure, très classique et tout à fait fondamentale étant donné son utilisation universelle, s'appelle une boucle. Mais contrairement aux boucles que nous avons vues au début, le nombre d'itérations est ici limité d'avance : N joue le rôle de compteur de passages sur les opérations à répéter; on lui donne une valeur initiale, puis on effectue le traitement à répéter pour les valeurs successives de N tant que la limite n'est pas dépassée.

Le programme B-4A nous prouve qu'on peut réaliser des boucles, de façon parfaitement satisfaisante, avec les instructions que nous connaissons déjà. Pourtant, étant donné l'importance des boucles, BASIC offre un jeu d'instructions spéciales qui permettent de les implanter encore plus facilement. C'est l'ensemble FOR... NEXT utilisé dans le programme B-4B :

```
1 REM  Programme B-4B
2 REM
10 FOR N=1 TO 10
20 INPUT "Devinez un nombre";A
25 E=100*ABS(A-3.25)/3.25
30 IF E<0.5 GOTO 60
40 PRINT "Erreur";INT(E*100)/100;"%"
45 NEXT N
50 PRINT "Je regrette. Vous avez perdu."
55 END
60 PRINT "Gagne' en";N+1;"coups"
```

On voit combien il est facile d'utiliser de telles boucles : il suffit d'encadrer les instructions à répéter (20 à 40 dans notre exemple) par FOR et NEXT :

– en tête, une instruction FOR, de la forme :

FOR N = valeur de départ TO valeur limite

ou, en français :

POUR N = valeur de départ JUSQU'A valeur limite

– à la fin, l'instruction NEXT N qui veut dire passer au suivant. Elle incorpore donc à la fois l'incrémentation de N et le test.

Évident, n'est-ce pas?

Exercice 4.6. – Imprimer une table des carrés et des racines carrées des entiers de 1 à 10.

L'instruction à répéter est d'imprimer sur la même ligne, un nombre N, son carré et sa racine, soit :

```
20 PRINT N;N*N;SQR(N)
```

Ceci est à faire pour toutes les valeurs de N de 1 à 10. Soit :

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

et l'on ne doit pas oublier de terminer par :

```
30 NEXT N
```

Exercice 4.7. – Implanter les instructions précédentes et faire exécuter. Que manque-t-il?

(Indication : ne concerne pas la boucle.)

EXTENSIONS DE FOR... NEXT

La forme que nous venons de voir n'est qu'un cas particulier de la forme plus générale :

FOR N = valeur de départ TO valeur limite STEP pas

STEP annonce le pas d'incrémentation du compteur. On met 2, par exemple, si l'on veut que N progresse de 2 en 2.

Exercice 4.8. – On veut faire la même table qu'à l'exercice 4.6., mais seulement pour les valeurs paires de N.

Si l'on ne met pas STEP et un pas, BASIC sous-entend un pas égal à 1.

Les valeurs des bornes peuvent être quelconques et le pas peut être négatif. Par exemple, si l'on voulait faire la même table que précédemment, mais en commençant par 10, puis 9, 8, etc., on écrirait :

```
10 FOR N=10 TO 1 STEP -1
```

On n'est pas obligé de mettre des constantes comme bornes : on peut mettre des variables, ou même n'importe quelle expression arithmétique. Mais les expressions sont évaluées une fois pour toutes lorsque l'on entre dans la boucle, même si l'exécution de la boucle fait évoluer les variables qui interviennent.

Ceci est utilisé surtout dans des écritures de la forme :

```
FOR I=1 TO N+1...
```

ou

```
FOR I=1 TO N STEP 2*K...
```

Rien n'oblige les paramètres à être entiers. Par exemple, pour étudier une fonction, on peut être amené à écrire :

```
FOR X=-3.5 TO 4.82 STEP 0.01
```

La valeur limite qu'on donne est la valeur qui ne sera pas dépassée pour l'exécution de la boucle ; le test se passe exactement comme sur l'ordinogramme du programme B-4A : le compteur est modifié et l'on teste s'il est toujours compris entre les bornes ; si oui, on exécute à nouveau la boucle, sinon, on a terminé et le compteur a une valeur hors des bornes.

Exercice 4.9. – Pour quelles valeurs du compteur sont exécutées les boucles suivantes et quelle est la valeur finale du compteur?

```
10 FOR I=1 TO 8.5 STEP 2
50 FOR M=10 TO 3.5 STEP -1
```

Le traitement à répéter dans une boucle peut lui-même contenir une boucle. On dit qu'on a des boucles imbriquées. Chacune doit avoir son compteur propre.

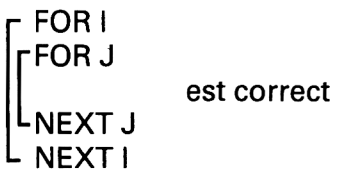
Exercice 4.10. – On veut dresser une table des nombres de 1 à 10 avec leurs carrés, mais avec 2 couples par ligne.

Une solution est :

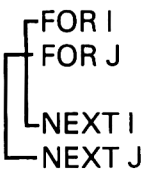
```
10 FOR I=1 TO 9 STEP 2
20 FOR J=0 TO 1
30 PRINT I+J;(I+J)*(I+J),
40 NEXT J :PRINT
50 NEXT I
```

Le PRINT en 50 assure un interligne double.

La deuxième boucle doit toujours être complètement contenue à l'intérieur de la première :



est correct



est incorrect

Remarque : si, en 30, on écrivait PRINT I+J;(I+J)^2, le programme serait ralenti (de façon imperceptible).

Pourquoi? Eh bien, c'est parce que l'interpréteur calcule x^y sous la forme $e^{y\log x}$ sans faire de cas particulier pour y=2, d'où des erreurs d'arrondi possibles et un temps plus long.

On voit là une des causes d'inefficacité des interpréteurs, alors qu'un programmeur en langage machine tient compte du cas particulier qu'il rencontre et remplace la puissance 2 par une multiplication, plus rapide.

On peut omettre de répéter la variable qui sert de compteur dans le NEXT. Ainsi, dans l'exercice 4.6., on aurait pu écrire 30 NEXT, et dans 4.10., 40 NEXT... 50 NEXT - et le problème de l'ordre ne se posait plus.

Un couple 40 NEXT J 50 NEXT I peut être remplacé par 45 NEXT J,I (attention à l'ordre).

Avec le couple FOR... NEXT, nous venons d'ajouter à notre arsenal un des outils les plus efficaces de BASIC. Il nous reste à voir deux autres éléments de base pour lesquels nous revenons à notre programme de jeu. Mais auparavant, nous voyons une autre sorte de boucles permises par le BASIC de l'Amstrad.

LA BOUCLE WHILE ... WEND (TANT QUE...)

La structure :

```
10 WHILE condition
20 ...
30 ...
40 WEND
50 ...
```

fait répéter les instructions 20 et 30 (par exemple) tant que la condition écrite en 10 est satisfaite ; lorsqu'elle ne l'est plus, on passe à 50.

A la différence de FOR, on ne connaît pas d'avance le nombre d'itérations qu'il faudra pour s'arrêter.

La "condition" a la même forme que dans IF. Il faut que les instructions entre WHILE et WEND aient des chances de faire évoluer la condition, sinon on ne s'arrêtera jamais. Si l'on arrive au WHILE alors que la condition est déjà fausse, il n'y aura aucune itération effectuée.

```
10 FOR I=1 TO 10 STEP K
:
50 NEXT
```

peut s'écrire :

```
5 I=1
10 WHILE I<=10
:
50 I=I+K
60 WEND
```

Il faut faire figurer l'incrémentation explicitement...

Le programme B−1 avec nombre illimité de tentatives peut s'écrire :

```
10 A=0: WHILE A<>3.25
20 INPUT A
30 WEND
40 PRINT "Gagné"
```

Bien sûr, on peut se passer de WHILE ... WEND puisqu'elle peut être simulée par d'autres instructions comme on l'a vu, mais elle offre une écriture extrêmement parlante. Son emploi à bon escient est donc recommandé.

L'HORLOGE TEMPS RÉEL

Ce qui manque dans notre jeu actuellement, c'est le "sport". Certes, le joueur a droit à un nombre de tentatives limité, mais ce serait beaucoup plus spectaculaire si le temps alloué pour deviner le bon nombre était limité.

L'Amstrad a ce qu'il faut pour cela. En effet, il possède une horloge temps réel, ce qui est intéressant pour sa catégorie de prix.

Mais qu'est-ce qu'une horloge temps réel et à quoi cela sert-il? Bien sûr, une horloge est faite pour donner l'heure, mais comment procède-t-on?

Dans le cas de l'Amstrad, l'horloge se comporte comme une case mémoire qu'on peut lire (c'est de cette façon qu'on obtient l'heure); mais cette case mémoire a un comportement un peu particulier : tous les trois centièmes de seconde, son contenu est augmenté de 1 par un processus indépendant du microprocesseur, qui fait intervenir un oscillateur, des divisions de fréquence et des interruptions, et dont nous n'avons pas à nous soucier.

Tout ce que nous devons savoir, c'est qu'il existe une variable (réservée) particulière, TIME, qui vaut 0 à la mise sous tension et qui, ensuite, a pour valeur le nombre de trois centièmes de seconde écoulés depuis la dernière mise sous tension.

Essayons le programme :

```
10 PRINT TIME
20 GOTO 10
```

Vous voyez un affichage qui n'arrête pas de varier, puisque le nombre est augmenté de 1 trois cents fois par seconde.

Les possibilités offertes par cette horloge temps réel sont très nombreuses. En effet, si l'on écrit :

```
10 T1=TIME
...
50 T2=TIME
```

alors T2−T1 est proportionnel au délai écoulé entre l'exécution de 10 et l'exécution de 50 : on a donc une mesure de ce délai (pour l'avoir en secondes, il suffit de diviser T2−T1 par 300). Cela peut permettre, entre autres, des mesures de performances.

On peut, au contraire, générer un délai : supposons qu'entre les instructions 10 et 20, on veuille respecter un délai d'une minute; on écrit :

```
10...
15 T=TIME
16 IF (TIME−T)<18000 GOTO 16
20...
```

En 15, on fixe l'instant de départ.

Ensuite, en 16, TIME représente le délai écoulé entre 15 et la présente exécution de 16 : il change à chaque fois car le temps passe. Tant que le délai est inférieur à 18000=3600 tierces = 60 secondes = 1 minute, on exécute à nouveau 16. Mais le délai avance sans cesse : il finira par être à 1 minute, et alors on passera en 20.

C'est donc cette horloge temps réel qui va nous servir pour limiter le temps alloué au joueur.

Comment? C'est tout simple. On va, au début du jeu, mettre l'heure à zéro :

```
5 T=TIME
```

puis, à chaque tentative du joueur, on va tester si le temps n'est pas dépassé :

```
15 IF (TIME-T)/300>120 GOTO 50
```

Exercice 4.11. – Quel est le temps alloué au joueur par l'instruction 15?

On arrive donc au programme B-5 :

```
1 REM  Programme B-5
2 REM
5 T=TIME
10 FOR N=1 TO 10
15 IF (TIME-T)/300>120 GOTO 50
20 INPUT "Devinez un nombre";A
25 E=100*ABS(A-3.25)/3.25
30 IF E<0.5 GOTO 60
40 PRINT "Erreur";INT(E*100)/100;"%"
45 NEXT N
50 PRINT "Je regrette. Vous avez perdu."
55 END
60 PRINT "Gagne' en";N+1;"coups et";INT((TIME-T)
/300);"secondes"
```

On a également incorporé à la ligne 60 l'impression du temps mis par le joueur pour trouver la solution.

Exercice 4.12. – Reconstituer l'ordinogramme du programme B-5.

Exercice 4.13. – Modifier l'instruction 60 pour imprimer le temps au 1/100ème de seconde (comme pour les compétitions de ski).

Il reste un petit perfectionnement à apporter : lorsqu'on imprime "Perdu", il serait bon de distinguer si c'est par dépassement de temps ou par trop grand nombre d'essais. C'est le but de l'exercice suivant.

Exercice 4.14. – Lorsque le joueur a perdu, imprimer la cause de l'échec, temps ou nombre d'essais.

INSTRUCTION STOP
TOUCHES CTRL STOP
COMMANDE CONT

Notre programme de jeu est maintenant arrivé à un bon niveau de complexité. Par conséquent, des difficultés peuvent se présenter pour en effectuer la mise au point. Comment l'Amstrad nous aide-t-il à les résoudre?

Le premier outil est constitué par les messages d'erreur imprimés par l'Amstrad en cas de situation anormale. Par exemple, si jamais vous appelez la fonction SQR avec un argument négatif, vous aurez le message :

Improper argument in numero d'instruction

puis Ready est affiché, ce qui indique que l'Amstrad est prêt à accepter une commande en mode direct.

La commande directe la plus judicieuse à entrer dans un cas semblable est PRINT certaines variables. En effet, lors d'un arrêt de cette sorte, toutes les variables du programme sont conservées, vous pouvez donc demander leur impression en mode direct. Par exemple, si notre racine carrée à argument négatif dépend d'une variable X, nous taperons ?X. Là, nous voyons que X n'a pas la valeur que nous escomptions. Nous pouvons alors demander le LIST de l'instruction qui calcule X. Nous voyons alors qu'il manque une opération et nous sommes prêts à corriger l'instruction.

Tel est le scénario habituel de correction des erreurs. Les messages d'erreur sont listés en annexe, avec une tentative d'analyse de leurs causes les plus fréquentes.

Lorsque le message est Syntax error in... (erreur de syntaxe en ...), la ligne incriminée est listée comme si l'on avait fait EDIT : on est donc tout prêt à la corriger.

Le procédé est différent s'il ne se produit aucune erreur suscitant un message alors que les résultats sont faux. Comment faire dans ce cas? A ce moment, on va subdiviser le programme en petites étapes, entre lesquelles on va insérer des instructions STOP. Par exemple, si un programme a deux étapes, la première de 10 à 50 et la seconde de 60 à 150, on intercalera un 55 STOP.

Lorsqu'on arrivera en 55, l'Amstrad imprimera :

```
Break in 55
Ready
```

et s'arrêtera.

Notons déjà que le fait d'obtenir une telle impression signifie que la première étape s'est déroulée jusqu'au bout. Correctement? Pour le savoir, puisque l'ordinateur est arrêté, il vous suffit de demander l'impression directe des variables stratégiques.

Supposons que tout soit correct. Nous voudrions maintenant exécuter la deuxième étape. Eh bien, pour cela, nous disposons de la commande CONT, qui veut dire "Continuez, maintenant que j'ai fait ce que je voulais lors de l'arrêt". Attention, vous ne pouvez employer CONT si, au cours de l'arrêt, vous avez modifié le programme.

Il y a encore un cas possible. Supposons que dans l'exemple ci-dessus, on n'obtienne jamais l'affichage de :

```
Break in 55
```

Cela veut dire que, lors de la première étape, le programme entre dans une boucle sans fin. Comment savoir où l'on en est? Il suffit d'appuyer deux fois sur la touche ESC. Cette combinaison simule une instruction STOP dans la ligne en cours d'exécution au moment où l'on appuie : on obtient le message Break in 25 (par exemple). On relance l'exécution par CONT. L'examen de quelques variables et un listing de la zone du programme indiquée par le Break in permettent, en principe, de dépister l'erreur. (*N.B.* – La touche ESC appuyée une seule fois interrompt l'exécution sans redonner le contrôle au mode direct.)

Un autre outil de dépannage est tout simplement d'ajouter, à intervalles réguliers, l'impression des principales variables; il suffira ensuite, lorsque le programme sera au point, de supprimer les instructions d'impression superflues.

LA "TRACE"

Un dernier outil, bien pratique, de dépannage est ce qu'on appelle la "trace". Lorsqu'elle est activée dans une portion de programme, à chaque fois que l'on passe sur une ligne BASIC, le numéro de cette ligne est systématiquement imprimé. On peut donc facilement suivre par où l'on passe, donc voir si un test s'effectue bien, ou si l'on boucle. On active la trace par TRON et on la désactive par TROFF.

Exemple

Le programme (idiot) suivant :

```
10 TRON
20 I=5
30 IF I<10 THEN 30
```

affichera [20] [30] [30] [30] [30]... indéfiniment.

Comme les impressions de trace troublent les impressions, on délimite les portions de programme soumises à la trace en les entourant de TRON... TROFF, qu'on supprimera une fois le programme au point.

Récapitulation

Ce chapitre nous a permis de voir les outils de base du programmeur :

- les instructions fondamentales IF et FOR;
- la manipulation de l'horloge temps réel de l'Amstrad ;
- quelques aides à la mise au point des programmes.

Nous sommes maintenant parés pour aborder les techniques élaborées de programmation.

Question : quelle est la différence entre STOP et END? – La seule différence est que STOP fait imprimer le message Break in... alors que END ne le fait pas.

On peut également reprendre par CONT après une instruction END.

CHAPITRE 5
PROGRAMMATION ÉVOLUÉE

AIDES A L'ÉCRITURE DES PROGRAMMES

Avant de passer à des programmes un peu plus longs que ceux que nous avons vus jusqu'à présent, dotons-nous de quelques outils qui nous faciliteront l'écriture - ou plutôt la frappe. Notons aussi que les micro-ordinateurs qui offrent ces facilités sont rares dans la gamme de prix de l'Amstrad.

AUTO

Passe en mode numérotation automatique, ce qui évite de taper les numéros. De la forme :

AUTO n1, n2

où n1 est le numéro de début et n2 l'intervalle entre numéros.

Exemple

AUTO 100,10 produira les numéros 100
110
120
etc.

Pour sortir du mode AUTO et revenir au mode direct, il faut faire ESC (un appui suffit).

RENUM

Renumérote les lignes du programme. De la forme :

RENUM n1, n2, n3

renumérote à partir de la ligne de n° n2 dans l'état actuel du programme; n2 devient n1 et l'intervalle entre les nouveaux numéros sera n3.

Bien entendu, les GOTO et autres sont remis à jour. *Exemple* : si 90 devient 200, un GOTO 90 devient GOTO 200.

Exemple

Avant :	100
	110
	112
	130
	150
après RENUM 120,112,10 on aura :	100
	110
	120
	130
	140

Cela sert surtout avant un MERGE.

DELETE

Supprime une série de lignes du programme.

De la forme DELETE n1−n2, avec les mêmes cas particuliers que LIST, elle supprime toutes les lignes de numéros compris entre n1 et n2.

Exemple

DELETE 100−200 supprime entre 100 et 200
DELETE −1000 supprime depuis le début jusqu'à 1000

Cela aussi peut être utile avant un MERGE.

MERGE

Permet de constituer un programme de morceaux rangés sur cassette et de réunir les morceaux. Elle est de la forme :

MERGE "nom"

et elle charge le programme nom à partir de la cassette (ou disquette), mais, à la différence de LOAD, le programme qui était déjà en mémoire ne sera pas supprimé. Le nouveau programme (lu sur cassette) viendra s'y réunir comme si on le tapait sur clavier. Les seules lignes de l'ancien programme qui seront perdues sont celles qui ont le même numéro qu'une ligne du nouveau programme : elles seront remplacées par les nouvelles. D'où l'intérêt d'un RENUM préalable dans certains cas.

Instructions DATA, READ et RESTORE

Notre programme de jeu va maintenant nous conduire à des techniques plus sophistiquées.

Supposons que l'on veuille jouer à plusieurs. Il nous faut donc maintenant une série de nombres à deviner. En effet, si le joueur n° 3 a vu que les deux précédents avaient à deviner 3.25, il n'aura pas trop à se creuser les méninges!

Le programme B-6A donne une solution. Pour simplifier, nous avons supprimé la limite sur le nombre d'essais autorisés, mais, bien sûr, nous avons laissé la limitation du temps.

Attention : si vous avez laissé en mémoire le précédent programme B-5, vous avez intérêt à effacer la mémoire avec la commande NEW avant de taper le programme ci-dessous, ou alors, faites un RENUM. Nous avons, en effet, renuméroté les lignes et la frappe superposée de B-6A sur B-5 donne une "salade" inintelligible. Cette renumérotation a pour but de redonner un peu d'air au programme pour permettre les futures adjonctions et modifications.

```
1 REM  Programme B-6A
2 REM
10 READ C
20 T=TIME
30 IF (TIME-T)/300>120 GOTO 90
40 INPUT "Quel nombre proposez-vous";A
50 E=100*ABS(A-C)/C
60 IF E<0.5 GOTO 110
70 PRINT "Erreur";INT(E*100)/100;"%"
80 GOTO 30
90 PRINT "Trop tard! Laissez la place au joueur
suivant"
100 GOTO 10
110 PRINT "Gagne' en";INT((TIME-T)/3)/100;"secon
des"
120 GOTO 10
200 DATA 3.25,7.63,2,121,449,0.075,18
210 DATA 5.8,210,78.31,901.5,31,4.18,2.7
```

Deux instructions nouvelles apparaissent dans cette version : READ et DATA.

DATA sert simplement à spécifier une liste de constantes séparées par des virgules; l'instruction DATA n'est considérée qu'en relation avec une instruction READ; sinon elle est "transparente" pour le programme.

On peut placer une instruction DATA n'importe où dans le programme; lorsque BASIC arrive dessus, il n'en fait rien et il passe à l'instruction suivante. Ainsi, 200 aurait pu être numéroté 55 et 210 aurait pu être numéroté 75, par exemple. L'exécution serait quand même passée directement de 50 à 60 et de 70 à 80. Les données prises n'auraient pas été altérées. Ce qui compte, c'est l'ordre des différentes instructions DATA et l'ordre des données à l'intérieur d'une même instruction DATA.

Au départ du programme (juste après le RUN), la première donnée du premier DATA sera prise lors du premier READ exécuté. Puis, au fur et à mesure de l'exécution des READ successifs, la deuxième donnée du premier DATA sera prise, puis la troisième, etc., puis la première donnée du deuxième DATA et ainsi de suite.

Dans notre exemple, comme il y a un READ à chaque nouveau joueur, les nombres successifs à chercher seraient 3.25, puis 7.63, puis 2, etc., soit 14 possibilités différentes.

Que se passe-t-il s'il y a plus de 14 joueurs? Eh bien, il y a une erreur : si l'on essaie un READ alors que la dernière donnée du dernier DATA a été "lue", l'Amstrad affiche le message DATA exhausted in...

L'instruction RESTORE nous permet de contourner l'obstacle : elle permet, en effet, de revenir au début des DATA. C'est-à-dire qu'après un RESTORE, un READ obtient de nouveau la première donnée du premier DATA, puis... etc. Ici, nous allons donc reparcourir la même série de nombres à deviner tous les 14 joueurs.

Pour cela, nous avons besoin d'une variable J qui va contenir le numéro de joueur. Quand ce numéro deviendra 14 ou divisible par 14, il faudra faire un RESTORE.

Mais, comment voit-on que X est divisible par Y? Très simple : si X est divisible par Y, le quotient X/Y est égal à INT(X/Y) puisqu'il est entier. D'où le programme B-6B :

```
1 REM  Programme B-6B
2 REM
10 J=J+1:PRINT "Joueur no.";J
20 READ C:T=TIME
30 IF (TIME-T)/300>120 GOTO 70
40 INPUT "Quel nombre proposez-vous";A
50 E=100*ABS(A-C)/C: IF E<0.5 GOTO 90
60 PRINT "Erreur";INT(E*100)/100;"%": GOTO 30
70 PRINT "Trop tard! Laissez la place au joueur
suivant"
75 IF INT(J/14)=J/14 THEN RESTORE
80 GOTO 10
90 PRINT "Gagne' en";INT((TIME-T)/3)/100;"second
es"
100 GOTO 75
200 DATA 3.25,7.63,2,121,449,0.075,18,5.8
210 DATA 210,78.31,901.5,31,4.18,2.7
```

Une variante supplémentaire, par rapport au programme B-6A, vient de ce qu'il y a plusieurs instructions à certaines lignes. On peut, en effet, mettre

plusieurs instructions par ligne à condition de les séparer par le caractère deux-points (:). Cela rend les programmes plus compacts, mais aussi moins lisibles.

Bien entendu, si la ligne "n" renferme plusieurs instructions, un GOTO n conduira à la première, pas au milieu de la ligne! Donc si une instruction doit être cible d'un GOTO ou d'un IF, elle doit être seule sur sa ligne, ou en tête de ligne.

Autre variante : le nombre 5.8 est passé du début du deuxième DATA à la fin du premier. Cela ne change absolument rien à l'ordre des données qui seront prises en compte.

Exercice 5.1. – Une autre méthode pour éviter l'épuisement des données serait d'ajouter, en fin de série, une donnée quelconque différente des nombres qu'on veut traiter, par exemple 99999, et que l'on teste : si on la trouve, on fait RESTORE. Réalisez une version du programme qui utilise cette méthode.

Une forme plus complète est RESTORE n où n est un numéro de ligne de DATA : on redémarre aux données de la ligne n.

DIM ET TABLEAUX

Nous abordons maintenant une importante notion qui va fortement augmenter la puissance de traitement mise à notre disposition.

Nous jouons toujours à plusieurs joueurs, que nous limitons à 14, par exemple. Ce que nous voulons, de surcroît, c'est qu'une fois que tous les joueurs ont effectué leur partie, un tableau récapitulatif des scores obtenus, en nombre de secondes, soit affiché.

Pour ce faire, il faut introduire une nouvelle variable : SC. On obtient alors le programme suivant :

```
1 REM  Programme B-7A
2 REM
10 FOR J=1 TO 14:PRINT "Joueur no.";J
20 READ C:T=TIME
30 SC=(TIME-T)/300: IF SC>120 GOTO 70
40 INPUT "Quel nombre proposez-vous";A
50 E=100*ABS(A-C)/C: IF E<0.5 GOTO 90
60 PRINT "Erreur";INT(E*100)/100;"%": GOTO 30
70 PRINT "Trop tard! Laissez la place au joueur
suivant"
75 IF INT(J/14)=J/14 THEN RESTORE
80 GOTO 100
90 PRINT "Gagne' en";INT(SC*100)/100;"secondes"
100 NEXT J
110 PRINT "Score =";SC: END
200 DATA 3.25,7.63,2,121,449,0.075,18,5.8
210 DATA 210,78.31,901.5,31,4.18,2.7
```

Mais cette solution n'est pas satisfaisante. En effet, ce programme n'imprimera jamais que le score du dernier joueur. Ce dont nous avons besoin, c'est un score pour chaque joueur, c'est-à-dire de 14 variables semblables, attachées chacune à un joueur J.

BASIC a un outil pour cela. Il permet, en effet, que SC soit une variable multiple dont SC(1) sera le premier élément, SC(2) le deuxième, etc.

Le numéro de l'élément voulu est mis entre parenthèses : il s'appelle l'indice. L'indice peut être une variable : SC(I) est le I^{e} élément, ou même une expression : SC(3*I+4).

Une telle variable multiple s'appelle un tableau. Il faut prévenir BASIC du fait qu'une variable est un tableau et annoncer le nombre d'éléments (cela pour réserver de la place en mémoire). Cela se fait par une instruction DIM. Pour notre exemple, nous avons : DIM SC(14). En fait, ici l'on réserve la place pour 15 éléments, car l'indice 0 est utilisable.

Bien entendu, l'instruction DIM doit être exécutée avant toute opération sur la variable concernée; en revanche, elle ne doit être exécutée qu'une fois.

L'instruction DIM n'est pas nécessaire tant que la valeur maximale de l'indice ne dépasse pas 10 : en effet, l'Amstrad réserve automatiquement la place pour 10. Si la dimension du tableau doit être plus petite que 10, l'instruction DIM est quand même utile car elle libère de la place.

Plusieurs tableaux peuvent être dimensionnés à l'aide d'une même instruction : DIM A(25),B(50).

La valeur maximale assignée à l'indice peut être une variable (qui vient d'être initialisée) : DIM A(N).

En utilisant ces propriétés, nous arrivons au programme B-7B :

```
1 REM  Programme B-7B
2 REM
5 DIM SC(14)
10 FOR J=1 TO 14:PRINT "Joueur no.";J
20 READ C:T=TIME
30 SC(J)=(TIME-T)/300: IF SC(J)>120 GOTO 70
40 INPUT "Quel nombre proposez-vous";A
50 E=100*ABS(A-C)/C: IF E<0.5 GOTO 90
60 PRINT "Erreur";INT(E*100)/100;"%": GOTO 30
70 PRINT "Trop tard! Laissez la place au joueur
suivant"
75 IF INT(J/14)=J/14 THEN RESTORE
80 GOTO 100
90 PRINT "Gagne' en";INT(SC(J)*100)/100;"seconde
s"
100 NEXT J
110 PRINT "Joueur  Score":PRINT
120 FOR I=1 TO 14
130 PRINT "  ";I;"  ";INT(SC(I)*100)/100:NEXT I
200 DATA 3.25,7.63,2,121,449,0.075,18,5.8
210 DATA 210,78.31,901.5,31,4.18,2.7
```

Nous laissons maintenant notre jeu de côté pendant quelques instants pour voir plusieurs compléments sur les tableaux. Vous avez tout intérêt à sauvegarder ce programme sur cassette grâce à l'ordre SAVE. Ensuite, vous pourrez effacer la mémoire par NEW et votre Amstrad sera disponible pour quelques exercices.

Garnissage d'un tableau

L'instruction d'entrée INPUT peut être mise dans une boucle du type :

```
FOR I=1 TO 10 : INPUT A(I) : NEXT I
```

On aimerait bien savoir à chaque instant quel élément doit être introduit et avoir des messages imprimés tels que :

```
A(1) ?
A(2) ?
etc.
```

Comme le message comprend un élément variable (la valeur de l'indice), la forme INPUT "Texte";... ne convient pas. Il faut utiliser un PRINT qui se termine par un point-virgule (;) pour taper la valeur sur la même ligne :

```
10 FOR I=1 TO 10
20 PRINT "A(";I;")";
30 INPUT A(I)
40 NEXT I
```

Somme et moyenne des éléments d'un tableau

Les éléments d'un tableau peuvent représenter les différentes observations statistiques d'une grandeur, par exemple les tailles des différents élèves d'une classe. La première opération statistique à effectuer sur une distribution est de calculer sa moyenne; pour cela, il faut d'abord en calculer la somme.

Pour calculer cette somme, nous utiliserons une variable S initialisée à 0 et à laquelle, dans une boucle, seront ajoutés successivement chacun des éléments :

```
10 DIM A(N)
20 REM NORMALEMENT INTERVIENT ICI LA LECTURE DES ELEMENTS
30 S=0
```

```
40 FOR I=1 TO N
50 S=S+A(I)
60 NEXT I
70 M=S/N
80 PRINT "Somme =" ;S; "Moyenne =" ;M
```

L'initialisation de S à 0 est effectuée en 30 ; cette initialisation est superflue en début de programme; elle peut être nécessaire si l'on arrive en 30 après avoir fait d'autres opérations.

En 20 apparaît une nouvelle instruction : REM (abréviation de remarque). Elle n'influe aucunement sur la marche du programme, mais elle permet d'y incorporer des commentaires explicatifs, ce qui est souvent utile (bien entendu, ces commentaires consomment de la place mémoire).

Exercice 5.2. – Calculer la variance de l'ensemble des éléments A ci-dessus.

$$V = \frac{\Sigma i(Ai-M)^2}{N-1}$$

Les tableaux sont particulièrement indiqués pour représenter des vecteurs d'un espace vectoriel sur K. Par exemple, A(1), A(2), A(3) seront les trois composantes du vecteur $\vec{A}$ dans l'espace à trois dimensions. On verra plus loin que BASIC permet aussi la manipulation de matrices.

Exercice 5.3. – Étant donné les deux vecteurs $\vec{U}$ et $\vec{V}$ d'un espace à N dimensions, calculer leur produit scalaire ($\Sigma u_i v_i$).

NOMBRES AU HASARD FONCTION RND

Nous sommes maintenant arrivés au moment de résoudre un des problèmes qui nous avait le plus préoccupés au début de notre jeu : comment faire pour que, même s'il triche, le joueur ne puisse trouver d'avance le nombre à deviner?

Le mieux est que l'ordinateur ne connaisse pas lui-même ce nombre à l'avance, c'est-à-dire qu'il le tire au sort au moment de l'utiliser.

Oui, mais comment un ordinateur dont le comportement doit être le plus déterministe et le plus prévisible possible peut-il donner des nombres aléatoires?

A priori, cela semble nuisible. Eh bien, et c'est paradoxal, il existe des algorithmes qui font appel à des calculs bien déterminés qui donnent ce que l'on appelle des séries pseudo-aléatoires, c'est-à-dire des suites de nombres bien déterminés, mais ayant des propriétés statistiques telles que l'on puisse considérer que tout se passe comme si les nombres avaient été tirés au hasard.

Dans ce contexte, le terme "nombre au hasard" ne peut s'appliquer à un nombre isolé, c'est seulement au niveau d'une suite (nombreuse) de nombres qu'il a un sens.

A quoi peuvent servir de telles suites? Elles servent pour des calculs de simulation dans lesquels il faut tenir compte de phénomènes aléatoires. Par exemple, supposons qu'on veuille simuler dix ans d'exploitation d'une propriété agricole. Un des éléments intervenants peut être la production de blé d'un certain champ; on sait que, quoi qu'il arrive, cette production est comprise entre 10 t (année aux conditions atmosphériques défavorables, mauvaises graines, etc.) et 15 t (année réunissant exceptionnellement tous les facteurs favorables).

Pour simuler les aléas dus à des causes mal connues, le mieux est, pour chaque année, de tirer au hasard un nombre entre 10 et 15 : ce sera le meilleur moyen d'obtenir, dans notre simulation sur dix ans, un certain nombre de bonnes années et de mauvaises années, représentatif de la réalité.

En fait, ici, on ne tirera pas un nombre réparti uniformément entre 10 et 15 : on cherchera à reproduire une loi de probabilité obtenue par observation ou répondant à un modèle.

On voit que les nombres au hasard peuvent être très utiles. L'Amstrad peut nous en fournir. Il suffit de faire Y=RND(X) pour obtenir un nombre au hasard compris entre 0 et 1.

Si X est >0, on génère différentes séquences pseudo-aléatoires : des appels successifs avec la même valeur de X>0, donnent les éléments successifs d'une même suite (le nombre obtenu change à chaque appel, mais il fait partie de la même suite). En changeant la valeur de X, on change de suite.

Le plus souvent, on utilisera des séries de RND(1), après un appel pour un argument négatif (qui choisit la série).

Le nombre obtenu étant compris entre 0 et 1, c'est-à-dire de la forme 0.232745, il faudra lui faire subir une transformation pour répondre à notre problème. D'une façon générale, il faut obtenir une série de nombres Y compris entre deux valeurs A et B. L'expression suivante permet d'obtenir ces nombres :

```
Y=A+(B-A)*RND(1)
```

Appliquons ceci dans le programme B-8 pour obtenir un nombre compris entre 0 et 100 :

```
1 REM  Programme B-8
2 REM
5 INPUT "Nombre de joueurs";N: DIM SC(N)
10 FOR J=1 TO N:PRINT "Joueur no.";J
20 C=1+99*RND(1):T=TIME
30 SC(J)=(TIME-T)/300: IF SC(J)>120 GOTO 70
40 INPUT "Quel nombre proposez-vous";A
50 E=100*ABS(A-C)/C: IF E<1 GOTO 90
60 PRINT "Erreur";INT(E*100)/100;"%": GOTO 30
70 PRINT "Trop tard! Laissez la place au joueur
suivant"
80 GOTO 100
90 PRINT "Gagne' en";INT(SC(J)*100)/100;"seconde
s"
95 SC(J)=1+INT((120-SC(J))/24)
100 NEXT J
110 PRINT "Joueur  Score":PRINT
120 FOR I=1 TO N
130 PRINT "  ";I;"    ";SC(I):NEXT I
```

Le changement le plus notable est donc dans l'instruction 20 où le nombre à chercher est maintenant tiré au hasard. On pourrait le rendre encore plus aléatoire (en effet, au départ du jeu, on aura toujours la même série de nombres) en l'indexant sur le temps, en écrivant, par exemple :

```
7 C=RND(-TIME)
```

ou

```
7 RANDOMIZE TIME
```

Dans ces conditions, selon le temps écoulé depuis la mise sous tension, une série différente sera générée. D'autres améliorations ont été introduites et ont légèrement facilité le jeu. Par exemple, on a mis la barre à 1 %, mais vous avez toute liberté de jouer sur cette barre et sur la limite de temps.

Le nombre N de joueurs est désormais variable. Le score est donné en nombre de points allant de 0 à 5 (0 si le joueur n'a pas trouvé dans les temps, 5 s'il a trouvé très vite).

Exercice 5.4. – Mesurer la qualité statistique du générateur de nombres au hasard de l'Amstrad.

Pour cela, nous allons tirer 1000 nombres au hasard compris entre −1 et +1. Calculons ensuite moyenne et variance (valeurs idéales 0 et 1/3) ainsi que les effectifs des 10 classes :

(−1,−4/5);(−4/5,−3/5)...(0,1/5)...(4/5,1) (valeur idéale 100).

Le programme suivant est une des solutions possibles :

```
1 REM  EX 5-4
2 REM
10 FOR I=1 TO 1000
20 N=(-1)+2*RND(1)
30 S=S+N:S2=S2+N↑2
40 J=1+INT((N+1)*5)
50 C(J)=C(J)+1
60 NEXT I
70 M=S/1000:V=(S2-1000*M↑2)/999
80 PRINT "Moyenne =";M,"Variance =";V
90 PRINT "Classes"
100 FOR I=1 TO 10:PRINT C(I);:NEXT
```

Attention, il ne se passe rien pendant 30 secondes. Soyez patients!

TABLEAUX MULTIDIMENSIONNÉS

Perfectionnons notre jeu en permettant à chaque joueur de disputer plusieurs parties et en affichant les scores de chacune de ces parties.

L'Amstrad permet de réaliser des tableaux à double entrée (dits aussi : tableaux à deux indices, ou à deux dimensions).

Pour ce faire, on utilise une instruction DIM de la forme :

DIM SC(NJ,NP)

s'il y a NJ joueurs et NP parties. Le score du joueur J à la partie P sera désigné par SC(J,P); d'où le programme B-9 :

```
1 REM  Programme B-9
2 REM
10 INPUT "Nombre de joueurs";NJ
20 INPUT "Combien de parties";NP
30 DIM SC(NJ,NP)
40 FOR P=1 TO NP: FOR J=1 TO NJ
50 PRINT "Joueur no. ";J;"   Partie ";P
60 C=1+99*RND(1):T=TIME
70 SC(J,P)=(TIME-T)/300: IF SC(J,P)>120 GOTO 110
80 INPUT "Quel nombre proposez-vous";A
90 E=100*ABS(A-C)/C: IF E<1 GOTO 130
100 PRINT "Erreur";INT(E*100)/100;"%": GOTO 70
110 PRINT "Trop tard! Laissez la place au joueur
 suivant"
120 SC(J,P)=0:GOTO 150
130 PRINT "Gagne' en";INT(SC(J,P)*100)/100;"seco
ndes"
140 SC(J,P)=1+INT((120-SC(J,P))/24)
150 NEXT J,P :PRINT:PRINT
160 PRINT "Joueur ";:FOR P=1 TO NP
170 PRINT "Partie";P;:NEXT P:PRINT:PRINT
180 FOR J=1 TO NJ:PRINT"  ";J;
190 FOR P=1 TO NP:PRINT"       ";SC(J,P);
200 NEXT P:PRINT:NEXT J
```

Ces tableaux rectangulaires s'appellent, en mathématiques, des matrices dont les éléments sont répartis en lignes et colonnes. Les matrices ont de nombreuses applications : celles-ci sont donc réalisables sur Amstrad.

Exercice 5.5. – Écrire un programme qui calcule le produit C de deux matrices A et B (rappel de la formule de définition :

$$C_{ij}=\sum_{k} a_{ij}.b_{kj}.$$

Ce qui vient d'être exposé concernant les tableaux se généralise : le nombre de dimensions peut être quelconque; à trois dimensions, les éléments sont répartis en plans, lignes et colonnes. La seule restriction formelle au nombre de dimensions est que l'instruction DIM doit tenir en 255 caractères. Mais d'autres limitations interviennent auparavant : la taille mémoire ne doit pas être dépassée.

ERASE

On ne doit passer qu'une fois sur le DIM d'un tableau. Si l'on veut changer la dimension d'un tableau sans avoir le diagnostic "Array already dimensioned", il faut d'abord le supprimer par ERASE nom du tableau. Bien sûr, toutes les valeurs sont perdues. La différence avec CLEAR est que cette dernière supprime toutes les variables alors que ERASE ne supprime que le(s) tableau(x) voulu(s).

MANIPULATION DES CHAÎNES DE CARACTÈRES

Il nous reste encore une amélioration à apporter à notre programme. En effet, il serait souhaitable que, dans l'impression des résultats, ce soit le nom de chaque joueur qui soit écrit et non son numéro.

Nous allons voir que cela est possible. Il est indispensable, en effet, que l'ordinateur permette de manipuler des textes ou des noms. Il faut bien, en gestion, manipuler le nom des clients!

L'Amstrad admet deux sortes de variables : les variables numériques, que nous connaissons déjà, et les variables alphanumériques, ou chaînes de caractères. Le nom d'une variable chaîne se forme comme celui d'une variable numérique, en ajoutant un $ à la fin :

A : variable numérique;
A$: variable alphanumérique.

Les variables distinctes A et A$ peuvent être utilisées concurremment dans le même programme.

Il est possible de former des tableaux de chaînes de caractères : DIM NOM$(N) réserve un tableau de N chaînes de caractères, chacun des éléments pouvant comporter un nombre variable de caractères.

Pour affecter une valeur à une chaîne de caractères, l'instruction la plus simple est l'affectation classique : A$="Bonjour".

Notez que la valeur attribuée est entourée de guillemets.

On peut aussi "lire" la variable au clavier par INPUT A$. Dans ce cas, il n'est pas nécessaire de mettre Bonjour entre guillemets car l'Amstrad s'attend à une chaîne de caractères.

On peut enfin utiliser READ et DATA. Dans le DATA, les chaînes sont normalement sans guillemets, sauf si elles contiennent un caractère "spécial" comme espace, virgule ou deux-points :

```
10 READ A$,B$
20 DATA BONJOUR, "AU REVOIR"
```

Nous sommes maintenant "parés" pour comprendre le programme B-10.

```
1 REM  Programme B-10
2 REM
10 INPUT "Nb de joueurs, nb de parties";NJ,NP
20 DIM NOM$(NJ),SC(NJ,NP)
30 FOR J=1 TO NJ:PRINT "Nom du joueur no.";J;
35 INPUT NOM$(J):NEXT J
40 FOR P=1 TO NP: FOR J=1 TO NJ
50 PRINT "Joueur no. ";J;"(";NOM$(J);")";"   Par
tie ";P
60 C=1+99*RND(1):T=TIME
70 SC(J,P)=(TIME-T)/300: IF SC(J,P)>120 GOTO 110
80 INPUT "Quel nombre proposez-vous";A
90 E=100*ABS(A-C)/C: IF E<1 GOTO 130
100 PRINT "Erreur";INT(E*100)/100;"%": GOTO 70
110 PRINT "Trop tard! Laissez la place au joueur
 suivant"
120 SC(J,P)=0:GOTO 150
130 PRINT "Gagne' en";INT(SC(J,P)*100)/100;"seco
ndes"
140 SC(J,P)=1+INT((120-SC(J,P))/24)
150 NEXT J,P :PRINT:PRINT
160 PRINT "Joueur ";:FOR P=1 TO NP
170 PRINT "Partie";P;:NEXT P:PRINT:PRINT
180 FOR J=1 TO NJ:PRINT NOM$(J);
190 FOR P=1 TO NP:PRINT"       ";SC(J,P);
200 NEXT P:PRINT:NEXT J
```

Dans le programme précédent, nous ne faisons que lire un nom, le ranger dans NOM$(J) pour le mémoriser et l'imprimer un peu plus tard. C'est souvent le seul traitement à effectuer sur les chaînes de caractères.

Mais, dans certains cas, on doit effectuer des traitements sur les chaînes, comme des comparaisons, des extractions, des conversions, etc. Nous voyons maintenant les opérations de ce genre disponibles sur l'Amstrad.

Comparaison. – Une instruction telle que IF A$=B$ GOTO... permet de comparer les deux chaînes A$ et B$. Les applications sont nombreuses : on peut, par exemple, vérifier si un mot appartient à un dictionnaire ou si un nom figure dans une liste de personnes autorisées... Une autre comparaison, telle que IF A$<B$... est également utile : en effet, le mot A$ est considéré comme inférieur à B$ s'il le précède dans l'ordre alphabétique, d'où un moyen de classer des listes de noms par ordre alphabétique. Les minuscules sont "supérieures" aux majuscules.

Concaténation. – L'opérateur + appliqué à deux chaînes de caractères produit leur juxtaposition :

"Bon"+"Jour" fournit Bonjour.

Tapez :

```
A$="NJ"
B$="UR"
C$="BO"+A$+"O"+B$
?C$
```

Là encore, vous obtenez Bonjour.

Chaîne vide. – C'est la chaîne formée de 0 caractère. A$= "" fournit à A$ la valeur "chaîne vide"; quelque soit X$, X$+A$ sera identique à X$.

Extraction de sous-chaînes. – L'Amstrad possède un certain nombre de fonctions permettant la "manipulation" des chaînes de caractères. Si le nom de la fonction se termine par $, son résultat est une chaîne de caractères; dans le cas contraire, ce résultat est un nombre.

– **LEN(X$)** : fournit la longueur (nombre de caractères) de la chaîne;

– **LEFT$(X$,N)** : fournit les N caractères les plus à gauche, extraits de la chaîne X$;

– **RIGHT$(X$,N)** : fournit les N caractères les plus à droite, extraits de la chaîne X$. Si N>LEN(X$), on obtient toute la chaîne (valable aussi pour LEFT$);

– **MID$** : extrait des caractères au milieu d'une chaîne; elle peut avoir deux ou trois arguments :

• **MID$(X$,K)** : fournit les caractères extraits de la chaîne X$ à partir de la position K. Si K>LEN(X$), on obtient la chaîne vide,

• **MID$(X$,K,N)** : fournit la sous-chaîne de N caractères extraits de X$ à partir du K^{e}. Si K>LEN(X$), on obtient la chaîne vide; si N spécifie plus de caractères qu'il n'en existe dans X$, on obtient tous les caractères de X$ à partir du K^{e};

– **STRING$** : forme une chaîne par répétition du même caractère;

• **STRING$(N,X$)** : fournit la chaîne formée de N fois le premier caractère de X$;

• **STRING$(N,X)** : fournit la chaîne formée de N fois le caractère de code X.

Exercice 5.6. – Remplacer le 5^{e} caractère de la chaîne X$ par la lettre A.

En fait, il suffit d'écrire MID$(X$,5,1)="A" car MID$ peut subir une affectation.

Exercice 5.7. – Vérifier si la chaîne A$ contient la sous-chaîne B$. Renvoyer dans la variable K, 0 si A$ ne la contient pas, et si elle la contient, la première position dans A$ où l'on trouve B$.

Par exemple, si B$="BRA", dans "ABRACADABRA", on trouve B$ en 2 et en 9 et l'on doit obtenir K=2. Avec "Bonjour", on doit obtenir K=0.

Cet exercice n'a qu'un intérêt scolaire, car la fonction INSTR donne la réponse : on écrirait K=INSTR(A$,B$).

– **INSTR(D,A$,B$)** : fournit le numéro de caractère où B$ commence dans A$. La réponse est 0 si B$ ne s'y trouve pas. La recherche commence au caractère n° D dans A$. Si D n'est pas spécifié, on fait comme si D=1.

– **LOWER$(A$)** : fournit une copie de A$, mais où toutes les majuscules ont été transformées en minuscules.

– **UPPER$(A$)** : fournit une copie de A$, mais où toutes les minuscules ont été transformées en majuscules.

Fonctions de conversion. – Les autres fonctions chaînes de caractères effectuent des conversions entre les nombres et leur représentation sous forme de chaînes de caractères ou de code ASCII (le code ASCII est le code de représentation interne choisi par la norme Amstrad et la plupart des micro-ordinateurs : chaque caractère est représenté par un motif binaire choisi, occupant un octet).

Il y a quatre fonctions de ce type :

– **ASC(X$)** : fournit la valeur en décimal de l'octet qui représente en ASCII le premier caractère de X$. *Exemple :* ?ASC("A") donne 65.

– **CHR$(K)** : fournit une chaîne de un caractère, le caractère dont K est le code ASCII. *Exemple :* ?CHR$(65) fait imprimer un A.

– **STR$(A)** : fournit la chaîne de caractères qui est la représentation en décimal du nombre A. Si A=3.5, STR$(A) est la chaîne : signe, chiffre 3, point, chiffre 5. Pour le signe, avec un nombre négatif, on a le signe –; avec un nombre positif, on a un espace. ?A et ?STR$(A) produisent la même impression à ceci près que ?A est suivi d'un mouvement de curseur à droite et non ?STR$(A). Observez la différence entre :

?"AA";A;"AAA" et ?"AA";STR$(A);"AAA"

Il y a néanmoins une différence importante : STR$(A) est une chaîne de caractères justiciable des opérations LEFT$, MID$, etc.

– **VAL(X$)** : fournit la valeur du nombre dont X$ est la représentation décimale. Les seuls caractères permis dans X$ sont les chiffres, le point, l'espace et + ou –. Si le premier caractère non blanc de X$ n'est pas l'un des caractères permis, on obtient 0. S'il y a des caractères permis puis des caractères non permis, on obtient le même résultat que s'il n'y avait que la première série de caractères permis : VAL("12 rue du 4 septembre") donne 12.

Exercice 5.8. – A est un entier positif. Trouvez son nombre de chiffres. Même question si A est entier quelconque.

Exercice 5.9.- Imprimez B en supprimant 3 décimales.

Fonctions de conversion de base. – L'Amstrad permet de manipuler des constantes entières exprimées dans d'autres systèmes de numération que le décimal. On peut utiliser les systèmes :

- binaire (base 2) *Exemple :* &B101 représente 5

et surtout

- hexadécimal (base 16) *Exemple :* &HFF représente 255

On voit que les constantes non décimales sont précédées d'un préfixe & plus initiale du système. La lettre peut être minuscule.

Le BASIC Amstrad comporte deux fonctions de conversion de base HEX$, et BIN$.

HEX$(X) donne la représentation hexadécimale du nombre X. *Exemple :* HEX$(255) donne FF, HEX$(1280) donne 500.

On remarque que la chaîne résultat n'a pas de préfixe. BIN$ se comporte comme HEX$, mais pour le système binaire.

Ces deux fonctions admettent un second argument qui est le nombre des chiffres voulus. Si cet argument est absent comme dans nos exemples précédents, ou insuffisant, la machine fournit juste les chiffres nécessaires. Si le nombre de chiffres est plus grand que nécessaire, on obtient des zéros à gauche.

Exemple

?BIN$ (5) donne 101.
?BIN$ (5,4) donne 0101.
?BIN$ (5,2) donne 101 aussi.

Et la conversion inverse? – Eh bien, en fait, VAL répond à la question. Par exemple, VAL("&HFF") donne le nombre 255. On voit qu'il faut le préfixe, cette fois. Donc, si H$ contient la représentation hexadécimale sans préfixe d'un nombre, pour avoir ce nombre en décimal, faites H=VAL("&H"+H$).

Remarque : si un nombre de la forme &H... est supérieur à &H8000 (32768), il est considéré comme négatif (entier 16 bits en complément à 2). Il faut ajouter 65536 pour avoir la valeur positive correspondante.

Exemple

HEX$ (40960) est A000 mais ?&HA000 donne −24576.
65536+VAL("&HA000") donne bien 40960.

Le 664 a, en plus, la fonction DEC$(X,F$) qui convertit X en chaîne décimale suivant le format F$:

DEC$(22.344, "# #.#") donne 22.3

Les formats qui servent aussi pour PRINT USING sortent du cadre de ce livre.

Récapitulation

Nous sommes maintenant arrivés à un état perfectionné de notre programme de jeu.

Au passage, nous avons découvert les principales possibilités de BASIC.

Les instructions décrites jusqu'ici sont en général valables pour tous les ordinateurs.

Nous allons voir maintenant des propriétés particulières de l'Amstrad qui, pour la plupart, ne se retrouvent pas sur les autre P.S.I., notamment les possibilités graphiques et sonores.

Mais, auparavant, nous voyons deux exercices essentiels.

Rappel. – Des solutions aux exercices sont proposées en annexe.

Exercice 5.10. – Calculez le score moyen de chaque joueur SM(J). Imprimez le nom et le score moyen du joueur qui a le score maximal. En cas d'ex-aequo, on prend le premier trouvé.

Il est particulièrement nécessaire ici de raisonner sur l'ordinogramme pour étudier ce problème du maximum. Ce problème très classique se pose dans d'innombrables applications. La fonction MAX que le BASIC de l'Amstrad est un des rares à posséder (MAX (3,4,1)=4) ne suffit pas.

Exercice 5.11. – Imprimez le classement des joueurs et non plus seulement le premier.

CHAPITRE 6
PROGRAMMES GRAPHIQUES

L'instruction la plus simple permettant de faire des dessins sur l'écran, nous la connaissons déjà. C'est tout simplement l'instruction PRINT.

PRINT "chaîne de caractères" fait imprimer une chaîne de caractères sur l'écran. Cela permet d'imprimer un texte, comme nous l'avons déjà vu. L'Amstrad possède tout un ensemble de caractères dits graphiques, spécialement adaptés au dessin, mais qui ne peuvent être, comme les autres, incorporés dans les guillemets d'une instruction PRINT. Ces caractères s'obtiennent à l'aide de la fonction CHR$ (*cf.* chapitre 5).

Ces caractères, ainsi que les valeurs de CHR$ correspondantes, apparaissent dans le tableau suivant.

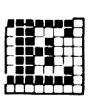
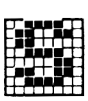
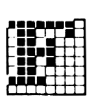
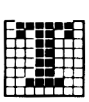
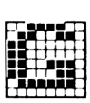
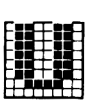
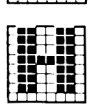
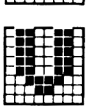
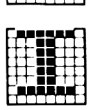
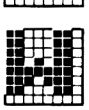
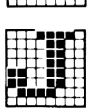
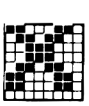
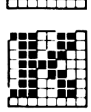
Les dessins de la colonne "Caractère" sont tirés du guide d'utilisation de l'Amstrad CPC 664.

Tableau 6.1. – Codes caractères

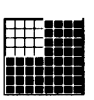
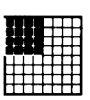
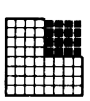
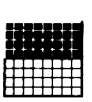
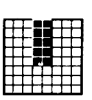
Caractère (3)	Touches (1) (4)	ASCII (2)	Hexa	Caractère	Touches	ASCII	Hexa
	ct @	0	0		ct C	3	3
	ct A	1	1		ct D	4	4
	ct B	2	2		ct E	5	5

Caractère	Touches	ASCII	Hexa	Caractère	Touches	ASCII	Hexa
	ct F	6	6		ct T	20	14
	ct G	7	7		ct U	21	15
	ct H	8	8		ct V	22	16
	ct I	9	9		ct W	23	17
	ct J	10	A		ct X	24	18
	ct K	11	B		ct Y	25	19
	ct L	12	C		ct Z	26	1A
	ct M	13	D			27	1B
	ct N	14	E			28	1C
	ct O	15	F			29	1D
	ct P	16	10			30	1E
	ct Q	17	11			31	1F
	ct R	18	12		espace	32	20
	ct S	19	13		sh 1	33	21

Caractère	Touches	ASCII	Hexa	Caractère	Touches	ASCII	Hexa
	sh 2	34	22		0	48	30
	sh 3	35	23		1	49	31
	sh 4	36	24		2	50	32
	sh 5	37	25		3	51	33
	sh 6	38	26		4	52	34
	sh 7	39	27		5	53	35
	sh 8	40	28		6	54	36
	sh 9	41	29		7	55	37
	sh :	42	2A		8	56	38
	sh ;	43	2B		9	57	39
	,	44	2C		:	58	3A
	-	45	2D		;	59	3B
	.	46	2E		sh ,	60	3C
	/	47	2F		sh -	61	3D

Caractère	Touches	ASCII	Hexa	Caractère	Touches	ASCII	Hexa
	sh .	62	3E		sh L	76	4C
	sh /	63	3F		sh M	77	4D
	@	64	40		sh N	78	4E
	sh A	65	41		sh O	79	4F
	sh B	66	42		sh P	80	50
	sh C	67	43		sh Q	81	51
	sh D	68	44		sh R	82	52
	sh E	69	45		sh S	83	53
	sh F	70	46		sh T	84	54
	sh G	71	47		sh U	85	55
	sh H	72	48		sh V	86	56
	sh I	73	49		sh W	87	57
	sh J	74	4A		sh X	88	58
	sh K	75	4B		sh Y	89	59

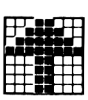
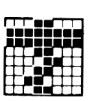
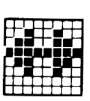
Caractère	Touches	ASCII	Hexa	Caractère	Touches	ASCII	Hexa
	sh Z	90	5A		H	104	68
	[	91	5B		I	105	69
	\	92	5C		J	106	6A
	]	93	5D		K	107	6B
	↑	94	5E		L	108	6C
	sh 0	95	5F		M	109	6D
	sh \	96	60		N	110	6E
	A	97	61		O	111	6F
	B	98	62		P	112	70
	C	99	63		Q	113	71
	D	100	64		R	114	72
	E	101	65		S	115	73
	F	102	66		T	116	74
	G	103	67		U	117	75

Caractère	Touches	ASCII	Hexa	Caractère	Touches	ASCII	Hexa
	V	118	76			132	84
	W	119	77			133	85
	X	120	78			134	86
	Y	121	79			135	87
	Z	122	7A			136	88
	sh [	123	7B			137	89
	sh @	124	7C			138	8A
	sh]	125	7D			139	8B
	ct 2	126	7E			140	8C
	(4)	127	7F			141	8D
		128	80			142	8E
		129	81			143	8F
		130	82			144	90
		131	83			145	91

Caractère	Touches	ASCII	Hexa	Caractère	Touches	ASCII	Hexa
		146	92			160	A0
		147	93			161	A1
		148	94			162	A2
		149	95			163	A3
		150	96			164	A4
		151	97			165	A5
		152	98			166	A6
		153	99			167	A7
		154	9A			168	A8
		155	9B			169	A9
		156	9C			170	AA
		157	9D			171	AB
		158	9E			172	AC
		159	9F			173	AD

Caractère	Touches	ASCII	Hexa	Caractère	Touches	ASCII	Hexa
		174	AE			188	BC
		175	AF			189	BD
		176	B0			190	BE
		177	B1			191	BF
		178	B2			192	C0
		179	B3			193	C1
		180	B4			194	C2
		181	B5			195	C3
		182	B6			196	C4
		183	B7			197	C5
		184	B8			198	C6
		185	B9			199	C7
		186	BA			200	C8
		187	BB			201	C9

Caractère	Touches	ASCII	Hexa	Caractère	Touches	ASCII	Hexa
		202	CA			216	D8
		203	CB			217	D9
		204	CC			218	DA
		205	CD			219	DB
		206	CE			220	DC
		207	CF			221	DD
		208	D0			222	DE
		209	D1			223	DF
		210	D2			224	E0
		211	D3			225	E1
		212	D4			226	E2
		213	D5			227	E3
		214	D6			228	E4
		215	D7			229	E5

Caractère	Touches	ASCII	Hexa	Caractère	Touches	ASCII	Hexa
		230	E6			243	F3
		231	E7			244	F4
		232	E8			245	F5
		233	E9			246	F6
		234	EA			247	F7
		235	EB			248	F8
		236	EC			249	F9
		237	ED			250	FA
		238	EE			251	FB
		239	EF			252	FC
		240	F0			253	FD
		241	F1			254	FE
		242	F2			255	FF

Notes sur le tableau 6.1.

(1) sh = SHIFT et ct = CTRL.

(2) On a indiqué dans cette colonne et la suivante le code ASCII et son équivalent hexadécimal. Un caractère peut toujours être obtenu par CHR$(ASCII) ou CHR$(&Hhexa) ou même CHR$(&Xbinaire).

(3) Les rôles des caractères de contrôle sont donnés dans le tableau 6.2.

(4) Lorsqu'aucune indication ne figure dans la colonne, c'est que le caractère ne peut s'obtenir avec une touche : il faut faire appel à CHR$.

Pour dessiner avec les caractères graphiques, nous allons incorporer les codes ASCII des caractères utilisés dans des DATA. On y inclura les retours-chariot (code 13 suivi de 10) qui font aller à la ligne.

A titre d'exemple, nous allons dessiner une maison.

Nous commençons chaque impression par un certain nombre d'espaces pour centrer le dessin. Le haut du toit commence par une barre inclinée (code 204), puis on a 5 traits horizontaux (code 208) et on termine par la barre inclinée de code 205. On commence cette ligne par 10 espaces (code 32).

Pour la seconde ligne du toit, on n'a que 9 espaces, puis la barre inclinée (204), 7 espaces et la barre (205).

Ligne suivante : 8 espaces, barre 204, 9 horizontales 210, barre 205.

Ligne suivante : 9 espaces, barre verticale 211, 7 espaces, barre verticale 209.

Ligne suivante : 9 espaces, barre verticale 211, barre 204, horizontale 208, barre 205, espace angle 150, T 158, angle 156, verticale 209.

Ligne suivante : 9 espaces, 2 fois 211, espace, 209, espace, 151, croix 159, 157, 209.

Ligne suivante : 9 espaces, espace, 209, espace, angle 147, 155, angle 153, 209.

Ligne suivante : 9 espaces, 2 fois 211, espace, 209, 4 espaces, 209.

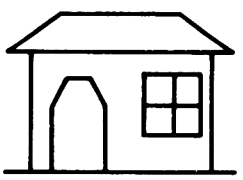
Ligne suivante : identique à la précédente.

Dernière ligne : 8 espaces, 11 fois horizontale 208.

D'où le programme C-1A, dont la partie exécutive est très simple : on imprime chaque caractère à mesure qu'il est lu par READ. On décèle la fin grâce à une donnée supplémentaire 9999.

```
1 REM  Programme C-1A
2 REM
10 READ A
20 IF A<>9999 THEN PRINT CHR$(A);:GOTO 10
100 DATA 32,32,32,32,32,32,32,32,32,32,204,208,2
08,208,208,208,205,13,10
110 DATA 32,32,32,32,32,32,32,32,32,204,32,32,32
,32,32,32,32,205,13,10
120 DATA 32,32,32,32,32,32,32,32,204,210,210,210
,210,210,210,210,210,205,13,10
130 DATA 32,32,32,32,32,32,32,32,32,211,32,32,32
,32,32,32,32,209,13,10
140 DATA 32,32,32,32,32,32,32,32,32,211,204,208,
205,32,150,158,156,209,13,10
150 DATA 32,32,32,32,32,32,32,32,32,211,211,32,2
09,32,151,159,157,209,13,10
160 DATA 32,32,32,32,32,32,32,32,32,211,211,32,2
09,32,147,155,153,209,13,10
170 DATA 32,32,32,32,32,32,32,32,32,211,211,32,2
09,32,32,32,32,209,13,10
180 DATA 32,32,32,32,32,32,32,32,32,211,211,32,2
09,32,32,32,32,209,13,10
190 DATA 32,32,32,32,32,32,32,32,208,208,208,208
,208,208,208,208,208,208,13,10
200 DATA 9999
```

Et voilà ! Nous obtenons une maison qui a l'aspect ci-dessous.



Les caractères répétés posent un problème : ils sont très fastidieux à taper. Au prix d'une légère complication du programme, on peut éviter les données répétées. Lorsqu'un caractère doit être répété, on fera précéder son code du facteur de répétition. Oui, mais comment distinguer un code d'un multiplicateur ? Simple : on donnera les multiplicateurs en négatif, d'où le programme C-1B. On en a profité au passage pour ajouter une cheminée à l'aide des caractères 140 et 131.

```
1 REM  Programme C-1B
2 REM
10 READ A: IF A=9999 GOTO 60
20 IF A>0 THEN PRINT CHR$(A);: GOTO 10
30 B=-A: READ A
40 FOR I=1 TO B: PRINT CHR$(A);
50 NEXT : GOTO 10
60 END
100 DATA -14,32,140,13,10,-10,32,204,-3,208,131,
208,205,13,10
110 DATA -9,32,204,-7,32,205,13,10
120 DATA -8,32,204,-9,210,205,13,10
130 DATA -9,32,211,-7,32,209,13,10
140 DATA -9,32,211,204,208,205,32,150,158,156,20
9,13,10
150 DATA -9,32,211,211,32,209,32,151,159,157,209
,13,10
160 DATA -9,32,211,211,32,209,32,147,155,153,209
,13,10
170 DATA -9,32,211,211,32,209,-4,32,209,13,10
180 DATA -9,32,211,211,32,209,-4,32,209,13,10

190 DATA -8,32,-11,208,13,10
200 DATA 9999
```

Vous devez être maintenant familiarisés avec une bonne partie des caractères graphiques de l'Amstrad. Essayez d'autres dessins, inspirés de celui-là, ou complétement différents.

Exercice 6.1. – Ajoutez un filet de fumée au-dessus de la cheminée.

POSITIONNEMENT SUR L'ÉCRAN

Le programme C-1 a plusieurs imperfections. La première est que le dessin apparaît n'importe où sur l'écran et qu'il n'est pas seul : ce que nous avions sur l'écran avant de faire RUN apparaît sur le haut de l'écran.

En mode programme, on utilise l'instruction CLS (les trois lettres C, L et S) pour vider l'écran. Pour positionner, c'est l'instruction :

LOCATE x, y où x est la colonne et y la ligne où l'on veut aller. Pour LOCATE, les lignes et colonnes sont numérotées à partir de 1.

Nous sommes maintenant en mesure de faire imprimer notre maison au milieu de l'écran, en commençant le programme par CLS et LOCATE. On termine par 60 GOTO 60 pour éviter l'impression de Ready. Vous pouvez ajouter des espaces dans les PRINT pour que la maison soit plus à droite.

D'où le programme C-2 :

```
1 REM  Programme C-2
2 REM
5 CLS:LOCATE 10,7:PRINT
10 READ A: IF A=9999 GOTO 60
20 IF A>0 THEN PRINT CHR$(A);: GOTO 10
30 B=-A: READ A
40 FOR I=1 TO B: PRINT CHR$(A);
50 NEXT : GOTO 10
60 GOTO 60
100 DATA -14,32,140,13,10,-10,32,204,-3,208,131,
208,205,13,10
110 DATA -9,32,204,-7,32,205,13,10
120 DATA -8,32,204,-9,210,205,13,10
130 DATA -9,32,211,-7,32,209,13,10
140 DATA -9,32,211,204,208,205,32,150,158,156,20
9,13,10
150 DATA -9,32,211,211,32,209,32,151,159,157,209
,13,10
160 DATA -9,32,211,211,32,209,32,147,155,153,209
,13,10
170 DATA -9,32,211,211,32,209,-4,32,209,13,10
180 DATA -9,32,211,211,32,209,-4,32,209,13,10

190 DATA -8,32,-11,208,13,10
200 DATA 9999
```

Le LOCATE est suivi d'un PRINT, sinon le premier caractère serait imprimé où le veut le LOCATE, mais les suivants en seraient séparés à cause des retours chariot.

Question : avec le 60, j'ai un programme qui ne s'arrête jamais. Comment en sortir? – On appuie deux fois sur ESC.

IMPRESSION DE L'HEURE

Au lieu de boucler sur le GOTO 60, on pourrait imprimer l'heure. Ceci suppose, bien sûr, d'avoir l'heure sous la forme heures minutes secondes, ce qui n'est pas le cas avec TIME.

Nous allons utiliser pour cela une possibilité très perfectionnée du BASIC Amstrad, la gestion des intervalles de temps.

Le BASIC Amstrad est capable, à chaque fois qu'un certain intervalle de temps s'est écoulé, d'interrompre l'instruction qu'il était en train d'exécuter et de sauter à l'instruction spécifiée. Il exécute les instructions qu'il trouve à partir de là jusqu'à ce qu'il rencontre RETURN. A ce moment, il revient là où il s'était interrompu. Une telle séquence exécutée jusqu'à RETURN s'appelle un sous-programme. Nous reviendrons sur cette importante notion.

Pour utiliser cette possibilité, il faut deux choses :

1. – Fournir le sous-programme de gestion de l'événement. Ici, l'événement c'est "temps écoulé = une seconde". Étant donné les variables S, M et H qui contiennent les secondes, minutes et heures, le principal traitement à faire est S=S+1. Plus précisément :

```
500 S=S+1 : IF S<60 THEN 530
510 S=0 : M=M+1 : IF M<60 THEN 530
520 M=0 : H=H+1
530 RETURN
```

Nous ne mettons rien au passage à 12 heures ou 24 heures mais vous pouvez le faire pour avoir une vraie horloge.

2. – Spécifier l'intervalle de temps voulu (ici, 1 seconde) et dire où aller (ici, 500) lorsqu'il est écoulé. Ici, l'on écrira :

```
10 EVERY 50 GOSUB 500
```

L'intervalle de temps se spécifie en cinquantièmes de seconde. On a, en fait, 4 chronos (n° 0 à 3, 0 par défaut) utilisables ainsi. Pour le chrono 2, on écrira :

```
10 EVERY 50,2 GOSUB 500
```

Voici alors un programme qui transforme votre Amstrad en horloge.

```
1 REM  Programme  C-3A
2 REM
10 INPUT "HEURE (h,m,s)";H,M,S:CLS
100 EVERY 50 GOSUB 500
110 LOCATE 13,12
120 PRINT H;M;S:GOTO 110
500 S=S+1:IF S<60 THEN 530
510 S=0:M=M+1:IF M<60 THEN 530
520 M=0:H=H+1
530 RETURN
```

Nous pouvons alors fusionner cela avec C-2 pour obtenir un coucou d'où le programme C-3B :

```
1 REM  Programme C-3B
2 REM
10 INPUT "HEURE (h,m,s)";H,M,S:CLS
20 LOCATE 1,7:PRINT
30 READ A: IF A=9999 GOTO 80
40 IF A>0 THEN PRINT CHR$(A);: GOTO 30
50 B=-A: READ A
60 FOR I=1 TO B: PRINT CHR$(A);
70 NEXT : GOTO 30
80 EVERY 50 GOSUB 500
90 LOCATE 9,21:PRINT H;M;S:GOTO 90
100 DATA -14,32,140,13,10,-10,32,204,-3,208,131,
208,205,13,10
110 DATA -9,32,204,-7,32,205,13,10
120 DATA -8,32,204,-9,210,205,13,10
130 DATA -9,32,211,-7,32,209,13,10
140 DATA -9,32,211,204,208,205,32,150,158,156,20
9,13,10
150 DATA -9,32,211,211,32,209,32,151,159,157,209
,13,10
160 DATA -9,32,211,211,32,209,32,147,155,153,209
,13,10
170 DATA -9,32,211,211,32,209,-4,32,209,13,10
180 DATA -9,32,211,211,32,209,-4,32,209,13,10

190 DATA -8,32,-11,208,13,10
200 DATA 9999
500 S=S+1:IF S<60 THEN 530
510 S=0:M=M+1:IF M<60 THEN 530
520 M=0:H=H+1
530 RETURN
```

CARACTÈRES DE CONTRÔLE

Les caractères de contrôle peuvent être incorporés dans une chaîne de caractères par CHR$(ASCII). Leur effet se produira lorsque la chaîne sera imprimée. Le tableau suivant résume les fonctions de contrôle et leurs codes.

Tableau 6.2. – Codes ASCII des caractères de contrôle

Code	Touche CTRL +	Graphique	Effet
0	@		Sans effet
* 1	A		En-tête de caractère graphique
2	B		Suppression curseur texte (664)
3	C		Rétablit curseur (664) (= CURSOR 1)
* 4	D		Établit mode écran (= MODE x)
* 5	E		Écrit un caractère sous le curseur graph.
6	F		Active écran texte
7	G		Sonnerie
8	H		Curseur à gauche
9	I		Curseur à droite
10	J		Aller à la ligne (curseur bas)
11	K		Curseur en haut
12	L		Vidage écran (= CLS)
13	M		Retour chariot / fin de message
* 14	N		Équivalent à PAPER
* 15	O		Équivalent à PEN
16	P		Supprime caractère sous curseur
17	Q		Vide le début de la ligne

Code	Touche CTRL +	Gra-phique	Effet
18	R		Vide la fin de la ligne
19	S		Vide l'écran jusqu'au curseur
20	T		Vide l'écran à partir du curseur
21	U		Désactive l'écran texte
* 22	V		Met mode transparent ou non (664)
* 23	W		Fixe le mode des graphiques (664)
24	X		Échange couleurs PEN et PAPER
* 25	Y		Équivalent à SYMBOL (9 paramètres)
* 26	Z		Équivalent à WINDOW (4 paramètres)
27	[		Sans effet
* 28	\		Équivalent à INK (3 paramètres)
* 29	]		Équivalent à BORDER (2 paramètres)
30	↑		Curseur en haut à gauche
* 31	0		Équivalent à LOCATE (2 paramètres)

Notes sur le tableau 6.2.

L'étoile devant certains codes signale les caractères de contrôle qui doivent être suivis de paramètres ; le nombre est indiqué s'il y en a plus d'un. Ces paramètres sont fournis sous forme de CHR$(valeur).

Exemple

? CHR$(4);CHR$(2) est équivalent à MODE 2.

Le paramètre peut aussi être fourni sous forme de caractère à imprimer, et c'est un caractère de contrôle si la valeur est inférieure à 31.

Exemple

? "CTRL D CTRL B" est aussi équivalent à MODE 2.

Les fonctions de contrôle s'obtiennent en exécutant un PRINT "CTRL ...". Lorsqu'on tape ce PRINT, il s'inscrit dans les guillemets un caractère graphique. Ces caractères forment le 3e colonne du tableau. Pour faire imprimer ces caractères (cela donne quelques possibilités graphiques supplémentaires), faire ? "CTRL A CTRL ...".

Exemple

? "CTRL G" fait retentir une sonnerie. Pour imprimer le dessin de la sonnette, faire ? "CTRL A CTRL G".

On peut, bien sûr, utiliser aussi la séquence CHR$(1)CHR$(...).

Exercice 6.2. – A titre d'exemple d'utilisation, nous allons écrire l'heure en haut à gauche de l'écran.

200 PRINT CHR$(30);H;M;S

où H, M et S sont obtenues comme dans le programme C-3A ou C-3B.

Exercice 6.3. – Afficher l'heure au milieu de l'écran.

On peut appliquer cela à une amélioration de notre dessin de maison. Nous allons constituer notre dessin en une chaîne de caractères comportant les caractères déjà vus et des mouvements de curseur pour les passages à la ligne. D'où le programme C-3C :

```
1 REM  Programme C-3C
2 REM
10 INPUT "HEURE (h,m,s)";H,M,S:CLS
20 P$=CHR$(31)+CHR$(9)+CHR$(9):M$=""
30 READ A: IF A=9999 GOTO 80
40 IF A>0 THEN M$=M$+CHR$(A): GOTO 30
50 B=-A: READ A
60 FOR I=1 TO B: M$=M$+CHR$(A)
70 NEXT : GOTO 30
80 PRINT P$+M$:EVERY 50 GOSUB 500
90 LOCATE 9,21:PRINT H;M;S:GOTO 90
100 DATA -6,32,140,10,-7,8,-2,32,204,-3,208,131,
208,205,10,-9,8
110 DATA 32,204,-7,32,205,10,-10,8
120 DATA 204,-9,210,205,10,-11,8
130 DATA 32,211,-7,32,209,10,-10,8
140 DATA 32,211,204,208,205,32,150,158,156,209,1
0,-10,8
150 DATA 32,211,211,32,209,32,151,159,157,209,10
,-10,8
160 DATA 32,211,211,32,209,32,147,155,153,209,10
,-10,8
170 DATA 32,211,211,32,209,-4,32,209,10,-10,8
180 DATA 32,211,211,32,209,-4,32,209,10,-10,8
190 DATA -11,208
200 DATA 9999
500 S=S+1:IF S<60 THEN 530
510 S=0:M=M+1:IF M<60 THEN 530
520 M=0:H=H+1
530 RETURN
```

On voit dans les DATA qu'un passage à la ligne se fait par CHR$(10) suivi de n fois curseur gauche (CHR$(8)). Pour le positionnement, on a mis dans P$ l'équivalent de LOCATE. LOCATE lui-même n'aurait pas marché car, la chaîne M$ étant longue, l'Amstrad qui ne sait pas qu'elle contient des passages à la ligne croit qu'elle ne tiendra pas sur la ligne et, donc, fait un retour-chariot intempestif avant d'imprimer M$.

On voit en tous cas l'intérêt de cette méthode : il suffit de changer P$ pour imprimer le dessin où l'on veut sur l'écran.

LA FONCTION INKEY$

Maintenant que nous savons écrire l'heure en un endroit fixe de l'écran, nous pouvons apporter une amélioration spectaculaire à notre programme de jeu du chapitre précédent.

Quelle angoisse pour le joueur de voir les secondes s'égréner sur l'écran pendant qu'il cherche le nombre à deviner!

Pour mieux voir l'essentiel, nous partons d'une version simplifiée du programme B-5/B-8 :

```
1 REM  Programme B-5B
2 REM
10 CLS:C=1+99*RND(1):T=TIME
20 IF TIME-T>18000 GOTO 70
30 LOCATE 1,1:PRINT "Nb. propose";
35 LOCATE 32,1:PRINT INT((TIME-T)/300)
40 INPUT A
50 E=100*ABS(A-C)/C:IF E<1 GOTO 80
60 PRINT "Erreur";INT(E);"%    ":GOTO 20
70 PRINT "Perdu!       ":GOTO 90
80 PRINT "Gagne'!      "
90 INPUT "On recommence";A$:IF A$="oui" GOTO 10
```

Les simplifications que nous avons apportées sont évidentes : nous avons supprimé toute gestion des différents joueurs, simplifié le traitement des erreurs et ramené le temps limite à une minute.

On remarque que la réponse se fait maintenant toujours au même endroit de l'écran. Les espaces en fin d'impression ont pour but de créer des sorties nettes.

La fin du programme mérite examen : on demande au joueur s'il veut recommencer et l'on analyse sa réponse; c'est très utilisé dans tous les programmes dits "interactifs".

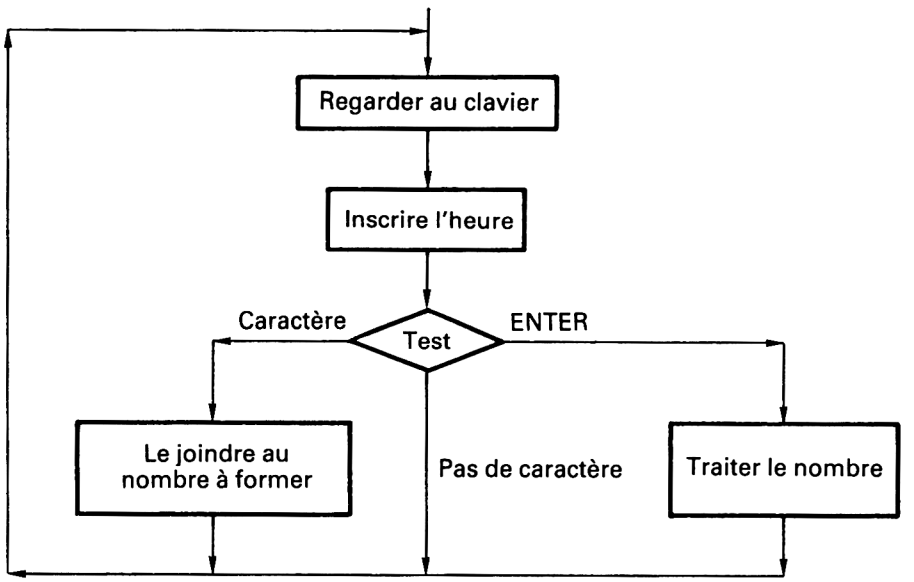
Nous avons mis le libellé "Nb. propose" (ligne 30) dans une instruction PRINT distincte de INPUT. C'est en effet entre les deux que se placera l'impression de l'heure. On inscrit dans la partie droite de la première ligne de l'écran le nombre de secondes écoulées à l'aide de l'instruction 35.

Essayons le programme ainsi obtenu.

Il ne fonctionne pas comme nous le voulons : on ne voit pas se dérouler le temps pendant que le joueur hésite à donner son nombre.

Tout vient de l'instruction INPUT. Lorsqu'une instruction INPUT s'exécute, le programme est bloqué jusqu'à l'appui sur la touche ENTER. Pendant ce temps, rien ne peut être fait.

Ce qu'il nous faudrait, c'est une instruction qui regarde si l'on a tapé un caractère sur le clavier et qui redonne le contrôle à l'utilisateur dans tous les cas. On pourrait alors obéir à l'ordinogramme ci-après :



Eh bien, l'Amstrad a ce qu'il faut pour cela : la fonction INKEY$. B$=INKEY$ fournit dans B$, ou bien le caractère sur lequel on a appuyé, ou bien la chaîne vide si l'on n'a pas tapé de caractère : juste ce qu'il nous fallait.

En appliquant exactement l'ordinogramme précédent, on obtient :

```
1 REM  Programme B-5C
2 REM
10 CLS:C=1+99*RND(1):T=TIME
20 LOCATE 1,1:PRINT "Nb. propose ?"
25 A$=""
30 IF TIME-T>18000 GOTO 70
33 B$=INKEY$
35 LOCATE 32,1:PRINT INT((TIME-T)/300)
37 IF B$="" GOTO 30
39 IF B$=CHR$(13) GOTO 45
41 A$=A$+B$: PRINT A$:GOTO 30
45 A=VAL(A$):PRINT
50 E=100*ABS(A-C)/C:IF E<1 GOTO 80
60 PRINT "Erreur";INT(E);"%    ":GOTO 20
70 PRINT "Perdu!       ":GOTO 90
80 PRINT "Gagne'!      "
90 INPUT "On recommence";A$:IF A$="oui" GOTO 10
```

Nous nous limiterons ici aux commentaires essentiels. Le changement d'ordre entre 20 et 30 est nécessaire pour que l'on passe à tout moment sur le test du temps, afin de bien assurer qu'on ne laisse qu'une minute.

On remarque, en 20, que l'on imprime explicitement le point d'interrogation : INKEY$ ne le fournit pas, au contraire d'INPUT. De même, avec INKEY$, on est obligé d'imprimer les caractères à mesure qu'on les trouve,d'où le PRINT en 41.

Les lignes 25, 33 et 41 construisent progressivement la chaîne A$ à partir d'une valeur initiale "chaîne vide". A chaque fois qu'un caractère B$ est disponible, il est concaténé à la chaîne A$ déjà obtenue : A$=A$+B$. Notez aussi, en 39, comment le caractère ENTER est testé; on ne pouvait pas faire : 39 IF B$="ENTER", car le ENTER termine impitoyablement la ligne. On aurait pu écrire également :

39 IF ASC(B$)=13...

Nous conseillons vivement d'adapter cette dernière version de B-5 à B-10 que nous avions obtenue au chapitre précédent : pratiquement, seuls les numéros d'instruction changent.

Nous voyons, en somme, que la fonction INKEY$ saisit les caractères un par un au clavier. Cela permet une meilleure interactivité entre l'utilisateur et la machine. Par exemple, à la fin de B-5C, on pourrait recommencer dès que l'utilisateur a tapé le O de OUI, sortir dès qu'il tape N, au lieu d'attendre O U I ENTER. Cela permet de réagir plus vite à la frappe de l'utilisateur : c'est nécessaire, en particulier lorsque, dans un jeu de bataille de chars, par exemple, l'appui sur une touche simule un tir.

Exercice 6.4. – Faites la modification nécessaire à la fin du programme B-5C.

Exercice 6.5. – Il est bien souvent utile d'insérer un point d'attente dans un programme : cela permet de fixer l'affichage pour que l'utilisateur en prenne connaissance. Très souvent, on décide que, pour continuer, il suffit que l'utilisateur appuie sur n'importe quelle touche. Implantez la séquence qui assure ce comportement.

Exercice 6.6. – Le programme C-3B ressemble beaucoup au fonctionnement d'un coucou. Pour que la ressemblance soit complète, il suffit qu'à x h 0 min 0 s, un coucou apparaisse dans la porte. Faites-le. Faites clignoter le coucou pendant qu'il apparaît.

CARTE DE FRANCE

Pour montrer notre maîtrise des graphiques, nous allons dessiner une silhouette de la France.

Pour un tel dessin, la seule méthode est la suivante : superposer une carte de France et un quadrillage géométriquement semblable à la "maille" de l'écran. Employez une grille voisine de la hauteur (la hauteur fait 5 % de plus que la largeur).

On aurait par exemple la grille suivante (les numéros en fin de ligne sont ceux des instructions DATA du programme que nous allons écrire) :

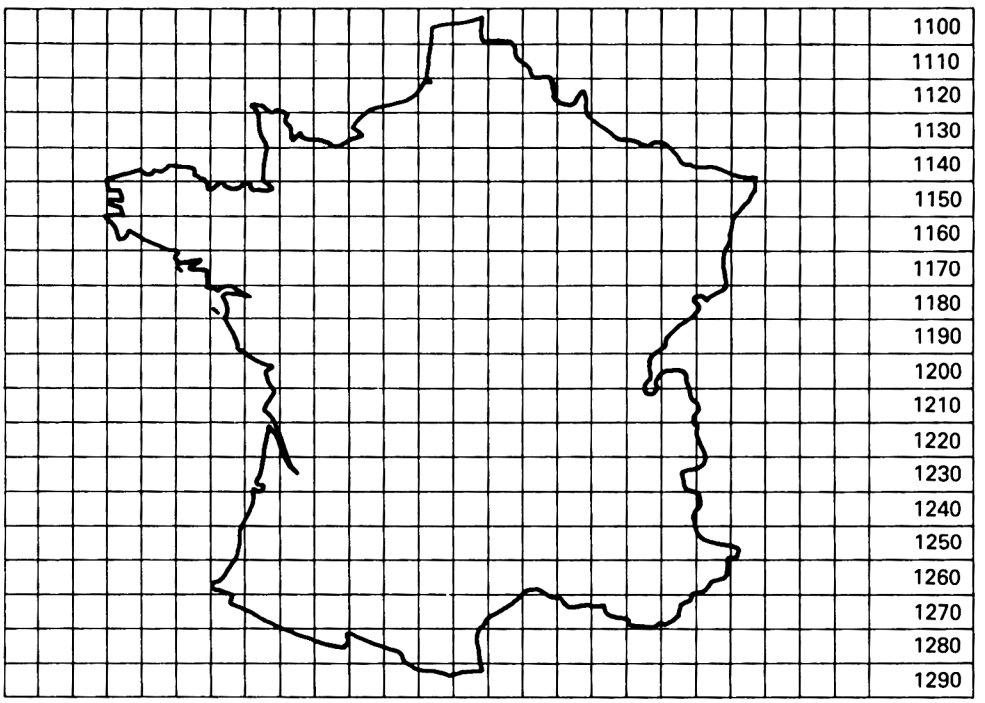


Figure 6.1.

Une fois ceci fait, il faut trouver le caractère - en principe graphique - qui convient le mieux à chaque carreau. La plupart sont des pleins (code 143) ou des espaces. Mais les autres? Voici, par exemple, les approximations que nous avons adoptées pour le Cotentin :

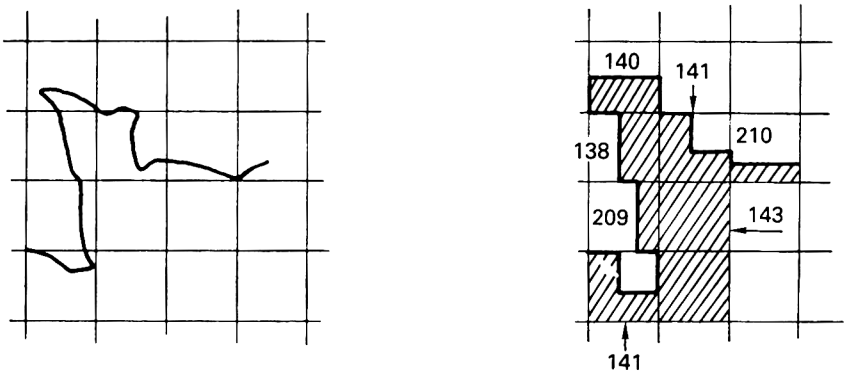


Figure 6.1.bis

D'où le programme C-5A :

```
1000 REM Programme C-5A
1001 REM Carte de France
1002 REM
1010 RESTORE 1100:CLS:PRINT
1020 READ CFA:IF CFA=9999 THEN 1060
1030 IF CFA>0 THEN PRINT CHR$(CFA);:GOTO 1020
1040 CFB=-CFA:READ CFA:FOR CFI=1 TO CFB
1050 PRINT CHR$(CFA);:NEXT:GOTO 1020
1060 GOTO 1060
1100 DATA -19,32,136,140,13,10
1110 DATA -19,32,138,143,143,132,13,10
1120 DATA -14,32,140,-3,32,210,142,-3,143,136,13,
,10
1130 DATA -14,32,138,141,210,139,-6,143,215,13,1
0
1140 DATA -10,32,210,95,140,32,209,-11,143,141,1
40,210,13,10
1150 DATA -10,32,139,-3,143,141,-13,143,212,13,1
0
1160 DATA -10,32,245,24,210,24,-16,143,211,13,10
1170 DATA -12,32,131,-15,143,13,10
1180 DATA -13,32,138,-13,143,208,13,10
1190 DATA -13,32,209,-12,143,212,13,10
1200 DATA -14,32,213,-10,143,24,209,24,13,10
1210 DATA -14,32,209,-10,143,141,143,132,13,10
1220 DATA -14,32,136,24,211,24,-11,143,211,13,10
1230 DATA -14,32,138,24,92,24,-10,143,143,129,13
,10
1240 DATA -14,32,24,211,24,-12,143,129,13,10
1250 DATA -14,32,-13,143,140,132,13,10
1260 DATA -13,32,214,-13,143,135,13,10
1270 DATA -13,32,130,139,-6,143,212,32,131,139,1
43,212,13,10
1280 DATA -15,32,208,131,208,139,143,143,13,10
1290 DATA -19,32,208,208,13,10
1300 DATA 9999
```

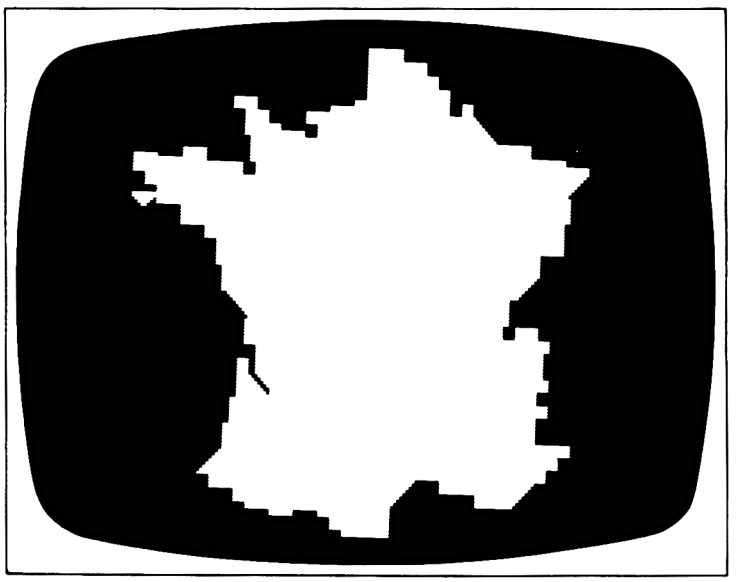


Photo 1

Ce programme utilise exactement les mêmes techniques que les programmes précédents, avec les multiplicateurs négatifs. Chaque ligne commence par une série d'espaces. La seule nouveauté sont les groupes 24, xxx, 24 : cela a pour effet de faire imprimer le caractère xxx en contraste inversé, ce qui enrichit encore le jeu de caractères disponibles.

Faisons RUN. Le résultat n'est qu'approximatif (voir *photo 1*). Mais n'y a-t-il pas moyen de l'améliorer ?

MODIFICATION DU GÉNÉRATEUR DE CARACTÈRES

Pour cela, il faudrait pouvoir spécifier - nous - le dessin des caractères. Mais déjà, comment la machine connaît-elle le dessin des caractères qu'elle a à afficher?

C'est relativement simple. Tout caractère est défini dans une grille 8 x 8 (cette grille est 8 fois plus fine dans chaque direction que la grille de notre carte de France). Sur chaque ligne de la grille, il y a 8 points susceptibles d'être allumés ou éteints. A chacun de ces points, on fait correspondre un bit, donc à chaque ligne de la grille correspond un octet et à chaque grille correspondent 8 octets.

Maintenant, un point allumé correspond à un bit 1, un point éteint correspond à un bit 0.

Pour définir un caractère, l'Amstrad dispose de l'instruction SYMBOL de la forme :

```
SYMBOL c,d1,d2,d3,d4,d5,d6,d7,d8
```

où c est le code ASCII du caractère que l'on redéfinit et les d dont les motifs qui définissent les points allumés ligne élémentaire par ligne élémentaire ; d1 est la ligne du haut, d8 celle du bas. Les d peuvent être définis en décimal ou en hexadécimal (&H...) ou en binaire(&X...), ce qui est le plus parlant.

Mais avant de donner l'instruction SYMBOL ou une série d'instructions SYMBOL, il faut fournir une instruction :

```
SYMBOL AFTER x
```

qui signifie "on va redéfinir un ou plusieurs caractères de codes >= x".

Exemple

La séquence :

```
SYMBOL AFTER 100
SYMBOL 100,&X00001100,&X00000011,&X00000001,
&X00001110,&X01110000,&X10000000,&X11000000,&X00110000
```

redéfinit le caractère d (code 100) comme un tortillon.

On aurait aussi pu écrire :

```
SYMBOL 100,12,3,1,14,224,128,192,48
```

Donc nous avons tout ce qu'il faut pour créer nos propres caractères.

Exercice 6.7. – L'exercice classique est de créer des lettres grecques. Mais l'Amstrad les a déjà. Remplacez le A par la lettre hébraïque ℵ (aleph). (Vous pouvez aussi prendre des caractères cyrilliques ou autres... ou des lettres accentuées.)

LA FRANCE EN HAUTE RÉSOLUTION

Nous sommes prêts maintenant à considérablement améliorer notre tracé de la silhouette de la France. Il nous suffit de définir sur mesure chaque caractère de la grille autre que les espaces ou les pleins. Cela marche parce que le nombre de caractères à redéfinir est limité : de toutes façons, il ne saurait dépasser 255. De fait, il y en a beaucoup moins (72) comme le montre la grille utilisée.

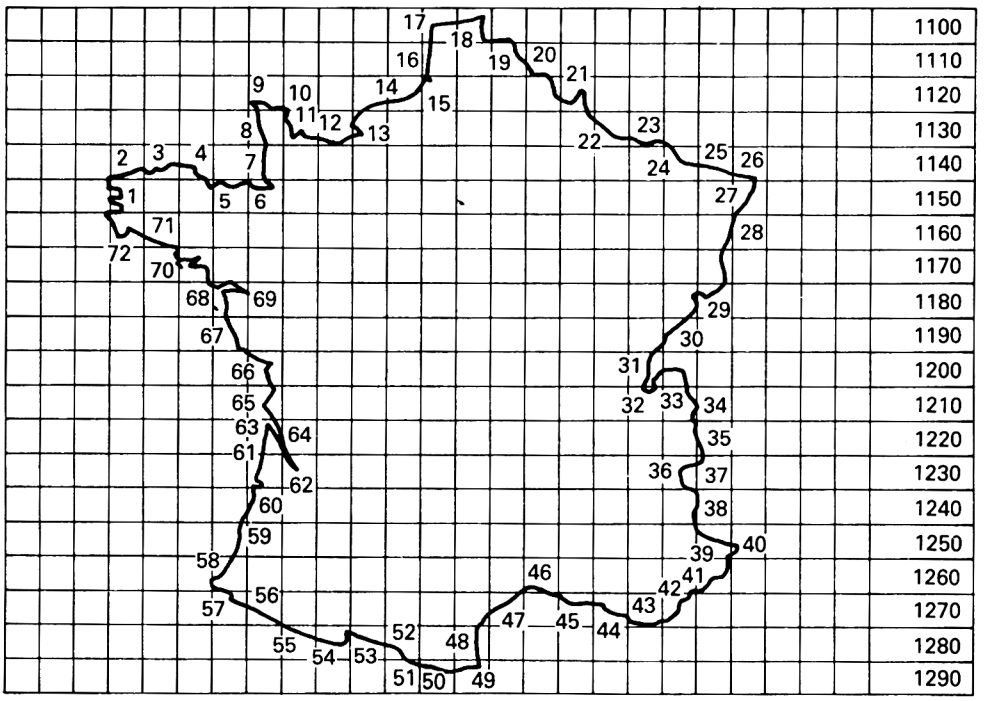
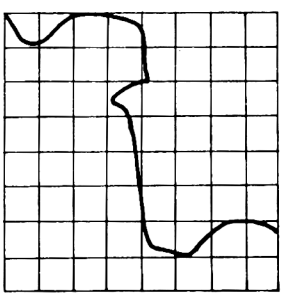


Figure 6.2.

Les numéros dans les carreaux frontière sont les codes écran des caractères redéfinis. On redéfinit les caractères de code 181 à 252. Le code est égal à 180 + le numéro marqué sur la figure 6.2. Les caractères espace et plein sont conservés. Les numéros à droite sont les numéros des instructions DATA. Les données sont écrites en décimal, ce qui est moins parlant mais plus court que le binaire.

Le reste est affaire de dessin. Par exemple pour le caractère 1 (181) on a les données 255,255,15,79,127,15,15,15. Pour le Cotentin, c'est plus délicat. Il faut utiliser une micro-grille :



d'où la définition du caractère 11 :48,240,224,240,240,240,253,255.

D'où le programme C-5B.

```
900 REM  Programme C-5B
901 REM  France Haute Resolution
902 REM
903 REM  Preparation caracteres
904 REM
910 SYMBOL AFTER 180
911 SYMBOL 181,255,255,15,79,127,15,15,15
912 SYMBOL 182,0,0,0,0,1,15,63,127
913 SYMBOL 183,0,0,0,0,1,3,155,255
914 SYMBOL 184,0,0,0,0,240,248,252,254
915 SYMBOL 185,24,25,255,255,255,255,255,255
916 SYMBOL 186,99,224,233,255,255,255,255,255
917 SYMBOL 187,3,3,7,7,7,7,7,7
918 SYMBOL 188,30,15,15,15,7,7,7,3
919 SYMBOL 189,0,0,0,0,0,0,56,30
920 SYMBOL 190,0,0,0,0,0,0,0,32
921 SYMBOL 191,48,240,224,240,240,240,253,255
922 SYMBOL 192,0,0,0,0,0,0,224,249
923 SYMBOL 193,3,7,15,63,31,15,31,255
924 SYMBOL 194,0,0,0,0,0,1,7,255
925 SYMBOL 195,15,15,63,127,127,255,255,255
926 SYMBOL 196,15,15,31,31,31,31,31,31
927 SYMBOL 197,0,0,0,0,15,15,15,15
928 SYMBOL 198,0,0,1,7,255,255,255,255
929 SYMBOL 199,252,254,254,255,255,255,255,255
930 SYMBOL 200,0,0,0,128,224,224,224,248
931 SYMBOL 201,0,0,129,131,135,207,255,254
932 SYMBOL 202,128,192,224,240,252,254,254,255
933 SYMBOL 203,0,0,0,0,0,0,112,249
934 SYMBOL 204,224,240,248,248,255,255,255,255
935 SYMBOL 205,0,0,0,0,128,222,255,255
936 SYMBOL 206,0,0,0,0,0,128,224,254
937 SYMBOL 207,255,254,252,248,248,240,224,192
938 SYMBOL 208,192,192,128,128,128,128,0,0
939 SYMBOL 209,255,255,158,8,0,192,128,0
940 SYMBOL 210,254,252,248,240,224,192,192,128
941 SYMBOL 211,255,254,252,252,252,252,252,248
942 SYMBOL 212,248,248,248,253,255,255,255,255
943 SYMBOL 213,0,0,0,0,126,255,255,255
944 SYMBOL 214,0,0,0,192,224,192,128,0
945 SYMBOL 215,0,128,192,192,224,224,224,240
946 SYMBOL 216,255,255,255,255,252,252,254,254
947 SYMBOL 217,240,240,224,128,0,0,0,0
948 SYMBOL 218,128,224,224,192,128,0,0,128
949 SYMBOL 219,128,128,192,240,252,255,255,255
950 SYMBOL 220,0,0,0,0,0,224,224,192
951 SYMBOL 221,255,255,255,255,248,240,240,224
952 SYMBOL 222,255,254,252,252,248,240,224,192
953 SYMBOL 223,255,255,255,255,255,127,127,127
954 SYMBOL 224,255,255,255,15,7,0,0,0
955 SYMBOL 225,63,63,31,0,0,0,0,0
956 SYMBOL 226,255,255,255,255,255,255,217,199
957 SYMBOL 227,255,254,252,224,192,128,0,0
958 SYMBOL 228,255,254,252,252,252,252,252,254
959 SYMBOL 229,254,255,255,128,0,0,0,0
```

```
960 SYMBOL 230,255,255,127,1,0,0,0,0
961 SYMBOL 231,7,1,0,0,0,0,0,0
962 SYMBOL 232,255,255,255,255,127,15,7,7
963 SYMBOL 233,255,63,31,1,0,0,0,0
964 SYMBOL 234,255,255,255,15,3,0,0,0
965 SYMBOL 235,63,15,3,0,0,0,0,0
966 SYMBOL 236,255,255,255,255,63,15,7,1
967 SYMBOL 237,7,7,7,3,0,0,0,0
968 SYMBOL 238,1,1,3,15,15,63,127,7
969 SYMBOL 239,63,127,127,255,255,255,255,255
970 SYMBOL 240,7,15,31,31,31,31,63,63
971 SYMBOL 241,7,7,15,15,15,15,15,3
972 SYMBOL 242,207,207,231,227,247,255,255,255
973 SYMBOL 243,0,0,2,7,7,7,7,7
974 SYMBOL 244,127,63,31,159,159,159,223,223
975 SYMBOL 245,1,0,1,3,7,3,1,1
976 SYMBOL 246,127,31,31,0,1,3,1,1
977 SYMBOL 247,15,7,7,3,3,1,1,0
978 SYMBOL 248,247,97,14,15,7,7,7,15
979 SYMBOL 249,255,255,63,255,255,255,255,255
980 SYMBOL 250,127,63,49,35,15,0,0,0
981 SYMBOL 251,255,255,255,255,255,127,31,7
982 SYMBOL 252,255,255,63,31,33,32,0,0
1000 REM
1001 REM Carte de France
1002 REM
1010 RESTORE 1100:CLS:PRINT
1020 READ CFA:IF CFA=9999 THEN 1060
1030 IF CFA>0 THEN PRINT CHR$(CFA);:GOTO 1020
1040 CFB=-CFA:READ CFA:FOR CFI=1 TO CFB
1050 PRINT CHR$(CFA);:NEXT:GOTO 1020
1060 GOTO 1060
1100 DATA -19,32,197,198,13,10
1110 DATA -19,32,196,143,199,200,13,10
1120 DATA -14,32,189,190,-2,32,194,195,-3,143,20
1,13,10
1130 DATA -14,32,188,191,192,193,-6,143,202,203,
13,10
1140 DATA -10,32,182,183,184,32,187,-11,143,204,
205,206,13,10
1150 DATA -10,32,181,143,143,185,186,-13,143,207
,13,10
1160 DATA -10,32,252,251,-16,143,208,13,10
1170 DATA -12,32,250,-15,143,13,10
1180 DATA -13,32,248,249,-12,143,209,13,10
1190 DATA -13,32,247,-12,143,210,13,10
1200 DATA -14,32,246,-10,143,211,213,13,10
1210 DATA -14,32,245,-10,143,212,143,214,13,10
1220 DATA -14,32,243,244,-11,143,215,13,10
1230 DATA -14,32,241,242,-10,143,216,217,13,10
1240 DATA -14,32,240,-12,143,218,13,10
1250 DATA -14,32,239,-12,143,219,220,13,10
1260 DATA -13,32,238,-8,143,226,-4,143,221,13,10

1270 DATA -13,32,237,236,-6,143,227,32,225,224,2
23,222,13,10
1280 DATA -15,32,235,234,233,232,143,228,13,10
1290 DATA -18,32,231,230,229,13,10
1300 DATA 9999
```

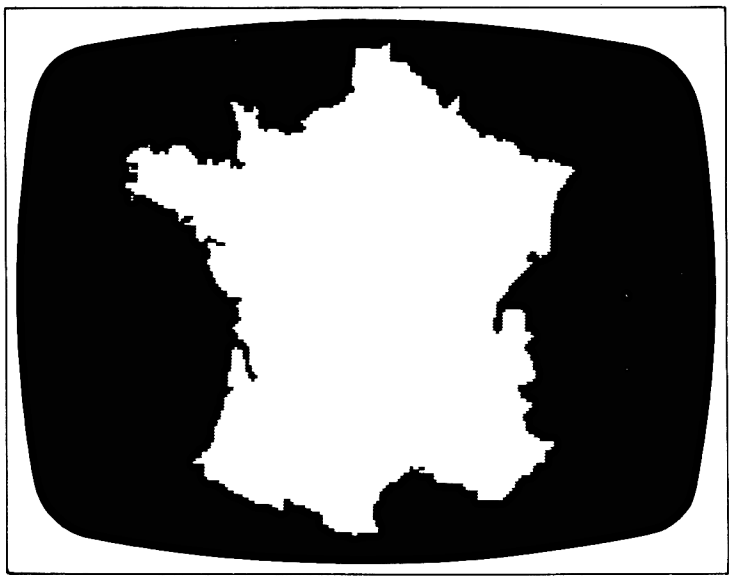
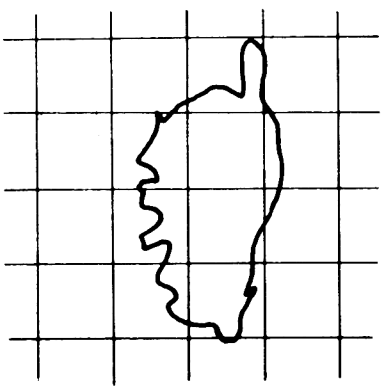


photo 2

Nous demandons à nos lecteurs corses de bien vouloir nous excuser de ne pas avoir fait figurer leur belle île sur notre carte, ainsi d'ailleurs qu'aux habitants des îles côtières qui n'apparaissent pas. Pour les dédommager, nous leur proposons l'exercice qui suit.

Exercice 6.8. – Cartographiez la Corse grâce à la grille ci-dessous.



Exercice 6.9. – Pour les lecteurs francophones, faites de même une carte de la Belgique,de la Suisse, du Québec, etc...

QUIZ GÉOGRAPHIQUE LES SOUS-PROGRAMMES

Maintenant que nous disposons d'une carte de France, nous allons pouvoir l'utiliser pour jouer à un jeu utile. Il va nous permettre de réviser nos connaissances géographiques. Il s'agit de reconnaître l'emplacement des principales villes de France. Il y a deux formes de jeu que nous implanterons toutes les deux.

A. – Le programme fait clignoter un carreau sur l'écran; vous disposez de 20 secondes pour taper le nom de la ville qui s'y trouve.

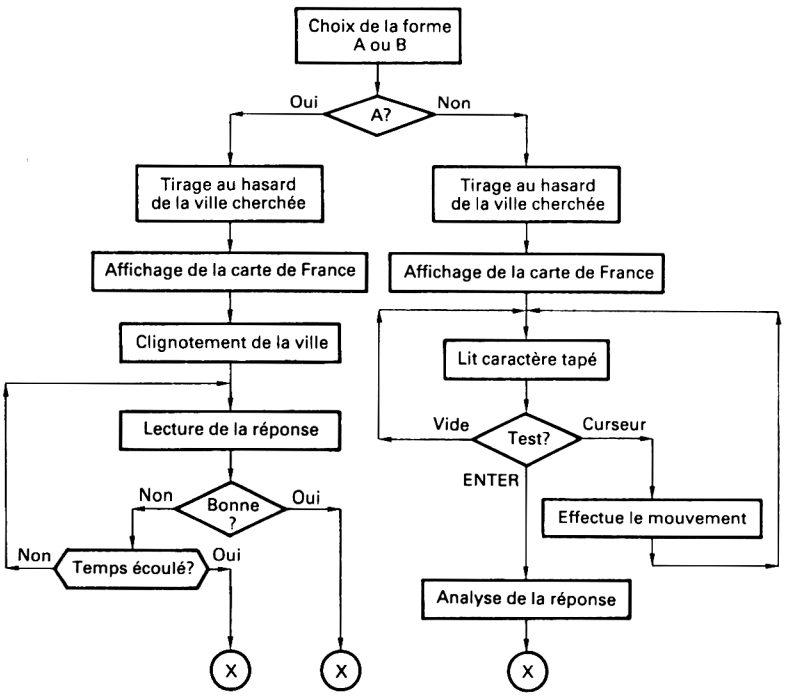
B. – Le programme donne un nom de ville. Vous devez, à l'aide des touches mouvement de curseur, amener le caractère  CHR$(134) sur l'emplacement de cette ville.

Tapez **ENTER** quand vous êtes satisfait de l'emplacement atteint. Bien sûr, vous disposez d'un temps limité.

Ceci est la base des programmes d'enseignement assisté par ordinateur, très répandus maintenant. Bien entendu, on peut les perfectionner en examinant les erreurs faites, en affichant des indications pour vous aider et en comptabilisant les résultats.

Nous avons vu précédemment tous les éléments qui permettent de le faire; nous nous bornerons ici aux solutions les plus simples.

Le programme obéit à l'ordinogramme ci-dessous :



Cet ordinogramme est simplifié. Ce qui apparaît fondamentalement, c'est que, en deux endroits, il faut exécuter la même opération : afficher la carte de France. Le programme de la carte est bien trop long pour le répéter en deux endroits.

Ce qui serait bon, c'est de disposer du mécanisme suivant *(cf. figure* ci-dessous).

```
100 ALLER A 1000
200 ALLER A 1000
1000 CARTE
1060 RETOUR
```

Le programme à répéter est écrit une seule fois, à partir de la ligne 1000, par exemple.

A chaque fois qu'on doit l'exécuter (ici en 100 et 200) une instruction qui ressemble à un GOTO fait sauter à 1000, où la séquence est exécutée.

La séquence se termine par une instruction qui veut dire "Retournez d'où vous venez". Étant venu de la ligne 100, retournez juste en dessous de 100 (parcours en trait plein). Étant venu de 200, retournez juste en dessous de 200 (parcours pointillé).

En somme, c'est un mécanisme de saut qui se "souvient" d'où il vient. Ce mécanisme existe en BASIC.

Le programme utilisé plusieurs fois s'appelle un sous-programme. L'instruction de saut vers le sous-programme est l'instruction d'appel GOSUB : 100 GOSUB 1000.

L'instruction de retour s'écrit tout simplement RETURN.

Le programme a donc la structure :

```
  :
  :
100 GOSUB 1000
  :
  :                      programme principal
200 GOSUB 1000
  :
  :
  :
1000
  :                      sous-programme (ici de
  :                      tracé de la carte de France)
1060 RETURN
```

Cette structure, que nous avions déjà aperçue à propos de EVERY, est extrêmement importante et puissante. Un programme peut appeler différents sous-programmes. Chaque sous-programme peut en appeler d'autres. Dans tous les cas, l'Amstrad s'y retrouvera pour effectuer les retours.

Cette simplification de notre travail étant acquise, nous avons besoin d'une liste de villes avec leur nom et, pour chacune, les coordonnées I (de 1 à 25) et J (de 1 à 40) qui la situent sur l'écran. Bien sûr, ces coordonnées seront approximatives, puisqu'au niveau de la maille-caractère.

Nous nous bornerons ici aux onze villes suivantes (cela évite les instructions DIM et rien n'empêche d'en ajouter ultérieurement).

Villes	I	J	N	Villes	I	J	N
Paris	7	21	0	Nantes	10	15	6
Marseille	19	25	1	Strasbourg	7	29	7
Lyon	14	24	2	St-Etienne	15	23	8
Toulouse	18	19	3	Le Havre	5	18	9
Nice	18	27	4	Lille	3	22	10
Bordeaux	15	16	5				

Les préliminaires du programme sont alors simples. Le tableau précédent est mis sous forme de DATA (lignes 10, 15, 20) qui sont ensuite lues (lignes 30 et 40).

Ensuite, c'est le choix du mode (lignes 50 et 60). (Pour une utilisation réelle du programme, il faudrait imprimer beaucoup plus d'explications que nous ne le faisons.)

Avant de taper ce programme, chargez votre programme carte de France et faites-y les quelques modifications qui suivent.

La première est de remplacer la ligne 1060 par :

```
1060 RETURN
```

Ensuite, on forme un autre sous-programme avec les instructions de redéfinition des caractères : en effet celui-ci n'a à être appelé qu'une fois et non pas à chaque fois qu'on trace la carte. D'où le 995 RETURN.

La dernière modification est la gestion du tableau EC$ qui contient une copie de l'écran. On en aura besoin pour rétablir l'affichage sous le curseur lorsqu'on promènera le caractère damier (voir instruction 520). Sur un 664, on n'en aurait pas eu besoin car la fonction COPYCHR$(#n) fournit le caractère sous le curseur. Il faut bien sûr avoir fait un LOCATE auparavant. L'argument n est le numéro de fenêtre (voir fin du chapitre 7) ; ici, ce serait 0.

Voici les lignes qui résument les principales modifications, entourées par 982 et 1100.

```
982 SYMBOL 252,255,255,63,31,33,32,0,0
985 DIM EC$(25,40)
990 FOR IP=1 TO 25 :FOR JP=1 TO 40 :EC$(IP,JP)="
 ":NEXT:NEXT
995 RETURN
1000 REM
1001 REM Carte de France
1002 REM
1010 RESTORE 1100:CLS:PRINT
1015 IC=2:JC=0
1020 READ CFA:IF CFA=9999 THEN 1060
1025 IF CFA=13 THEN IC=IC+1:JC=0:PRINT CHR$(CFA)
;:READ CFA:PRINT CHR$(CFA);:GOTO 1020
1030 IF CFA>0 THEN PRINT CHR$(CFA);:JC=JC+1:EC$(
IC,JC)=CHR$(CFA):GOTO 1020
1040 CFB=-CFA:READ CFA:FOR CFI=1 TO CFB
1050 PRINT CHR$(CFA);:JC=JC+1:EC$(IC,JC)=CHR$(CF
A):NEXT:GOTO 1020
1060 RETURN
1100 DATA -19,32,197,198,13,10
```

Voici maintenant le début du Quiz.

```
1 REM  Programme C-6  Quiz Geographique
2 REM
10 DATA Paris,Marseille,Lyon,Toulouse,Nice,Borde
aux,Nantes
15 DATA Strasbourg,Saint Etienne,Le Havre,Lille
20 DATA 7,21,19,25,14,24,18,19,18,27,15,16,10,15
,7,29,15,23,5,18,3,22
30 FOR V=0 TO 10:READ NOM$(V):NEXT
40 FOR V=0 TO 10:READ JJ(V),II(V):NEXT
45 GOSUB 900
50 CLS:INPUT "A ou B";X$
55 V=INT(11*RND(1)):T=TIME
60 IF X$<>"A" AND X$<>"a" GOTO 500
```

En fait, le tirage de la ville voulue et l'initialisation du temps sont communs et seront en 55 :

```
55 V=INT(11*RND(1)):T=TIME
```

- *Jeu A*

On se trouve après 70.

```
70 PRINT "Jeu A: Vous tapez le nom de la ville q
ui clignote puis 'ENTER'"
72 FOR I=1 TO 2000:NEXT
75 I=II(V):J=JJ(V)
80 GOSUB 1000:A$="":Q=134:K=1
85 IF (TIME-T)>10000 GOTO 170
90 Q=271-Q:LOCATE I,J:PRINT CHR$(Q);:LOCATE K,22
95 B$=INKEY$:IF B$="" GOTO 85
100 IF B$=CHR$(13) GOTO 150
105 K=K+1:A$=A$+B$
110 LOCATE K,22:PRINT B$;:GOTO 85
150 IF A$=NOM$(V) GOTO 180
160 LOCATE 2,23:PRINT "NON ":FOR I=1 TO 2000:NEX
T:GOTO 75
170 CLS:PRINT "Perdu": GOTO 190
180 CLS:PRINT "Gagne'"
190 INPUT "On recommence";A$
195 IF A$="oui" GOTO 50
200 END
```

Le programme est bâti avec des matériaux déjà vus, notamment la construction caractère par caractère du nom A$ de la ville proposée par l'élève (instructions 80 à 110).

En 72 et en 160, délais pour fixer l'affichage.

En 75, I et J sont les coordonnées sur l'écran de la ville cherchée. Il y aura à faire attention au fait que l'ordre I, J correspond naturellement à l'ordre lignes colonnes (c'est l'ordre suivi pour le tableau EC$) alors qu'il est inversé dans l'instruction LOCATE.

De 80 à 90, on imprime alternativement  (code 134) et  (code 137), ce qui fait un effet de clignotement. Remarquez comment on passe de 134 à 137 et vice versa en soustrayant de 271 qui est la somme des deux.

En 105-110, on imprime le caractère tapé. Si c'est **ENTER**, on passe à l'analyse du nom proposé par le joueur. Les noms doivent être tapés avec la même orthographe que dans les DATA (première lettre majuscule).

Le reste du programme ne devrait pas faire de difficulté. Nous passons donc au jeu B.

- *Jeu B*

Nous sommes cette fois après la ligne 500.

```
500 PRINT "Jeu B: Avec les touches curseur, vous
"
505 PRINT "amenez le damier sur la ville de  ";N
OM$(V):FOR I=1 TO 6000:NEXT
510 GOSUB 1000
515 I=1:J=1
520 CC$=EC$(J,I):LOCATE I,J:PRINT CHR$(134);:LOC
ATE I,J
525 IF (TIME-T)>40000 GOTO 170
530 A$=INKEY$:IF A$="" GOTO 525
535 IF A$=CHR$(13) GOTO 570
540 IF A$=CHR$(243) THEN I=I+1
545 IF A$=CHR$(242) THEN I=I-1
550 IF A$=CHR$(241) THEN J=J+1
555 IF A$=CHR$(240) THEN J=J-1
560 PRINT CC$;:GOTO 520
570 IF I=II(V) AND J=JJ(V) GOTO 180
575 LOCATE 2,23:PRINT "NON";:LOCATE I,J
580 FOR R=1 TO 1000:NEXT:GOTO 560
```

I et J, initialisés à 1, indiquent l'emplacement du damier (code 134).

Là encore, pour des raisons d'interactivité, on utilise INKEY$ pour examiner la touche enfoncée. Selon la touche utilisée, le mouvement est réalisé, c'est-à-dire que I ou J est modifié. La touche **ENTER** est testée par CHR$ puisqu'on ne peut la mettre entre guillemets. De même, pour les touches curseur, les codes ne sont pas les codes contrôle déjà vus : il faut employer les codes clavier 240 à 243 cités dans la notice constructeur à propos du schéma du clavier. Le mouvement est effectivement réalisé lignes 560 et 520 ; Le caractère situé en I, J est mémorisé dans CC$. Les nouvelles coordonnées sont alors calculées, et le damier est déplacé à cette nouvelle position, alors que CC$ est réécrit à l'ancienne position.

Lorsque l'utilisateur tape **ENTER**, le I et le J atteints sont alors l'emplacement proposé. Celui-ci est comparé aux données des tableaux II et JJ.

Notez l'utilisation de la variable R pour le délai en 580 afin de ne pas modifier I utilisé par ailleurs!

L'ordre des instructions en 510-515 est tel que lorsque la réponse est NON, une autre tentative est possible. On reprend alors à la position proposée à l'essai précédent, ce qui permet d'atteindre le but par de faibles corrections successives.

Pour l'ensemble des essais, le temps est limité à 2 min (contre 20 s dans le jeu A).

Récapitulation

Ce programme, qui est l'archétype des programmes utilisés en enseignement assisté par ordinateur, n'exploite que les techniques simples qui ont été progressivement introduites au cours du jeu "Devinez un nombre" et pour la programmation du dessin d'une maison.

La seule technique nouvelle introduite ici - et elle est très importante - est celle des sous-programmes.

Avec un simple GOSUB, le tracé de la carte de France est effectué chaque fois que nécessaire dans le programme.

La fonction INKEY$ apporte également une bonne interactivité dans le dialogue homme-machine, essentielle dans cette application.

Nous ne proposons pas d'exercice sur ce programme, construit pas à pas, mais nous ne saurions trop encourager le lecteur à s'exercer à y apporter toutes les modifications qu'il jugerait bon d'essayer. Vous pouvez notamment vous inspirer des dernières versions du programme B pour gérer les "scores" obtenus par différents élèves. Vous pouvez aussi changer les délais et ajouter des villes (dans ce cas, attention à DIM...).

Remarque. – Le sous-programme carte de France a été écrit à partir de 900, de façon à en permettre l'utilisation dans plusieurs programmes. Vous le stockez sur cassette, lorsque vous souhaitez l'incorporer dans un programme à écrire, vous le chargez en premier. Puis vous écrivez votre programme dans les premiers numéros.

Cette façon de procéder, qui nous a évité la recopie fastidieuse d'une partie de programme, est généralisable grâce à l'emploi de MERGE.

LA COULEUR

Nous ne pouvons clore ce chapitre sans voir les effets de couleur. Mais, ils dépendent des modes d'affichage.

Il y a en effet trois modes d'affichage, de numéros 0, 1 et 2. On passe dans le mode m par MODE m.

Dans chaque mode, on définit un écran texte et un écran graphique.

Dans le mode 0, on a 200 lignes élémentaires de 160 points élémentaires (ces points s'appellent des "pixels"). Chaque pixel peut avoir une couleur

parmi 16. Il peut y avoir au maximum 16 couleurs simultanément sur l'écran. Pour le texte, comme chaque caractère est défini dans une matrice 8×8, on a 25 lignes de 20 caractères.

En mode 1, qui est le mode de la mise sous tension, on a 200 ×u 320 pixels qui peuvent avoir 4 couleurs différentes. Pour le texte, on a 25 lignes de 40 caractères.

En mode 2, on a la résolution maximale : 200 × 640 pixels ou 25 lignes de 80 caractères. Cela exige une grande qualité d'affichage, inaccessible avec les téléviseurs ordinaires ; c'est probablement pour cela qu'Amstrad livre des moniteurs. La haute résolution se paie : en mode 2, il n'y a plus que deux couleurs différentes possibles sur l'écran : on dit qu'elles forment la couleur des points éteints (le fond) et la couleur des points allumés (le tracé).

Deux choses à noter :

1. – A toutes ces couleurs, il s'en ajoute une, la couleur du cadre de l'écran. Elle est déterminée par l'instruction BORDER de la forme :

BORDER c

où c est un code de couleur donné par le tableau 6.3.

Exemple

BORDER 5 produit un cadre bleu clair. On peut spécifier deux codes couleur, et, dans ce cas, l'on a clignotement entre deux couleurs spécifiées (à éviter si vous tenez à la santé de vos yeux !).

Exemple

Essayez BORDER 0,1.

2. – Lorsque vous avez un moniteur noir et blanc, les couleurs sont remplacées par des dégradés de gris du plus foncé (code 0) au plus clair (code 26).

Les valeurs des codes de couleur correspondent au tableau :

Tableau 6.3. – Codes de couleur

0	noir	9	vert	18	vert cru
1	bleu	10	turquoise	19	vert citron
2	bleu vif	11	bleu ciel clair	20	turquoise pastel
3	rouge brun	12	jaune moutarde	21	comme 18
4	violet	13	blanc	22	comme 19
5	bleu ciel	14	comme 11	23	comme 20
6	rouge	15	orangé	24	jaune
7	fuchsia	16	rose	25	jaune clair
8	parme	17	parme clair	26	blanc clair

Comme on a 16, 4 ou 2 couleurs possibles selon le mode, alors qu'il y a au choix 23 couleurs (théoriquement 27, mais certaines se distinguent vraiment difficilement - ce qui peut, d'ailleurs, dépendre du réglage de votre moniteur), les couleurs utilisées seront déterminées en deux temps.

On définira d'abord une "palette" de 16, 4 ou 2 couleurs selon le mode à l'aide d'autant d'instructions INK de la forme :

INK n,c

où c est un code de couleur (cf. *tableau 6.3*) et n est un numéro de couleur de 0 à 15, 3 ou 1 selon le mode : si vous donnez un numéro trop grand pour le mode, l'Amstrad fera un modulo. INK n,c signifie "la couleur n° n est la couleur de code c", par exemple INK 1,6 signifie que la couleur n° 1 sera le rouge.

On peut spécifier deux codes couleur : la couleur sera alors le clignotement entre les deux couleurs spécifiées.

En mode 2, on aura donc besoin de 2 instructions INK. Il en faudra 4 en mode 1 et 16 en mode 0. Rassurez-vous, si vous n'avez pas spécifié toutes les "encres", il y a des valeurs par défaut (qui se trouvent dans le manuel constructeur).

La détermination réelle des couleurs se fait par référence aux numéros de couleur définis par les instructions INK. On a deux instructions pour cela :

PEN n qui impose que tous les tracés ou textes subséquents seront de la couleur n° n

et

PAPER n qui impose les fonds de couleur n° n

Attention : PEN 1 ne signifie pas du tout "lettres jaunes". Il a ce sens parce qu'on a exécuté une instruction INK 1,24 (en fait, c'est le système qui l'a fournie par défaut).

Bien sûr, si vous donnez le même numéro de couleur dans PAPER et PEN, vous ne verrez rien ! PAPER impose le fond des prochains caractères écrits ; pour imposer une couleur à tout le fond de l'écran, il faut que votre PAPER soit suivi d'un CLS.

Exercice 6.9. (macabre) – Préparez un faire-part de décès.

Exercice 6.10. – On voudrait la France en orangé sur fond bleu océan.

Exercice 6.11. – Faire sur l'écran un pavage aléatoire de rectangles de couleurs.

CHAPITRE 7
COURBES – GRAPHIQUES HAUTE RÉSOLUTION DESSINS ANIMÉS

Avant d'attaquer les dessins animés proprement dits, nous allons réaliser un programme "sérieux", très important pour ses applications : le tracé de la courbe représentative d'une fonction.

COURBE REPRÉSENTATIVE D'UNE FONCTION INSTRUCTION DEF FN

Le programme n'est, en fait, pas très compliqué. Nous allons tracer la courbe de la fonction exponentielle y=exp(x), (e^x) pour x allant de 0 à 1. La courbe sera tracée point par point. Temporairement, l'axe des x sera vertical, l'axe des y sera horizontal. Disposant de 25 lignes sur l'écran, on pourra représenter 24 points.

Sur la ligne i sera matérialisé le point correspondant à x=i/24. L'axe des abscisses est ici vertical, orienté de gauche à droite. y peut, lui, varier de 1 à 40. Sur une ligne donnée, une étoile sera imprimée dans la colonne dont le numéro est proportionnel à la valeur de y pour le x qui correspond à la ligne.

Il faut donc appliquer un facteur d'échelle à la fonction, pour qu'elle varie entre 1 et 40 afin de couvrir tout l'écran.

Pour la fonction exp, on emploiera :

$$y = \frac{39 \cdot (\exp(x) - 1)}{e - 1} + 1.$$

La formule générale pour une fonction f(x) dont le maximum et le minimum dans l'intervalle de variation étudié sont respectivement max et min serait :

$$y = \frac{39 \cdot (f(x) - mini)}{maxi - mini}.$$

Pour imprimer la courbe, il suffit donc de réaliser une boucle FOR x = début TO fin STEP intervalle et pour chaque point mettre une étoile dans la colonne INT(y).

BASIC dispose de deux fonctions simples pour cela :

- PRINT TAB(N) : qui positionne à la colonne N;
- PRINT SPC(N) : qui fait imprimer N espaces donc positionne en N+1.

On peut aussi se positionner à l'aide de LOCATE.

Le programme à réaliser en découle directement :

```
1 REM  Programme D-1A
2 REM
10 FOR X=0 TO 1 STEP 1/24
20 N=1+INT(39*(EXP(X)-1)/(EXP(1)-1))
30 PRINT SPC(N);"*"
40 NEXT
```

Bien entendu, un programme véritablement opérationnel devrait offrir une meilleure présentation : titre, dessin des axes, coordonnées, etc.

Vous avez toutes les connaissances utiles pour le faire et nous vous suggérons de vous y exercer.

Exercice 7.1. – Tracez deux courbes sur le même graphique, par exemple sin(x) et cos(x) pour x de 0 à 2* PI.

Indication : ? CHR$(11) fait remonter l'impression à la ligne précédente.

Nous traitons l'exercice dans le texte, vu les difficultés qu'il pose et les solutions apportées.

Compte tenu de l'indication donnée, une solution simple est :

```
1 REM  Ex 7-1A
2 REM
10 CLS
20 FOR X=0 TO 2*PI STEP 2*PI/22
30 N1=1+INT(38*(SIN(X)+1)/2)
35 N2=1+INT(38*(COS(X)+1)/2)
40 PRINT SPC(N1);"*"
45 PRINT CHR$(11);SPC(N2);"+"
50 NEXT X
```

La 2e courbe s'inscrit avec des +. On a réduit l'intervalle d'écriture à 23 lignes (d'où le 22 à l'instruction 20) et l'intervalle de variation à 39 (d'où le 38 en 30 et 35) pour que les courbes tiennent bien dans l'écran.

Mais il y a un défaut : les SPC(N2) de la 2e courbe effacent la première courbe là où y1<y2. Il faudrait tester pour imprimer le plus grand d'abord, ce qui complique le programme.

En fait, au lieu d'espaces, il faudrait des curseurs à droite. De même que PRINT CHR$(11) équivaut à un curseur en haut, on a les codes suivants :

CHR$(...)	Mouvement
28	curseur à gauche
29	curseur à droite
10	curseur bas
11	curseur haut
12	vidage écran
13	retour chariot
30	retour en haut à gauche de l'écran (sans effacer)
32	espace

Nous allons donc nous constituer une chaîne de N curseurs à droite par STRING$(N,9).

D'où la seconde version :

```
1 REM  Ex 7-1B
2 REM
10 CLS:H$=CHR$(11)
20 FOR X=0 TO 2*PI STEP 2*PI/22
30 N1=1+INT(38*(SIN(X)+1)/2)
35 N2=1+INT(38*(COS(X)+1)/2)
40 PRINT STRING$(N1,9);"*"
45 PRINT H$;STRING$(N2,9);"+"
50 NEXT X
```

INSTRUCTION DEF FN

Il peut arriver de devoir tracer la courbe d'une fonction qui n'est pas exactement une des fonctions de la bibliothèque, mais une combinaison de plusieurs d'entre elles.

Cette combinaison peut intervenir plusieurs fois dans le programme. Il est alors pénible de la réécrire à chaque fois.

Il existe une instruction qui permet de résoudre ce problème : c'est une instruction qui permet à l'utilisateur de définir ses propres fonctions, qui vont alors s'ajouter aux fonctions de bibliothèque.

Exemples

```
10 DEF FNHARM(X)=(A*X+B)/(C*X+D)
20 DEF FNA(X)=2*SIN(X)+COS(2*X)
30 DEF FNF(T)=EXP(-T ↑ 2/2)
40 DEF FNG(X)=1+39*(FNF(X)-MINI)/(MAXI-MINI)
```

Le nom de la fonction définie suit FN; il obéit aux règles habituelles applicables aux noms de variables.

Un appel à la fonction FNHARM définie en 10 pourrait être :

```
100 Z=1+FNHARM(3.5)
```

L'expression donnée dans la définition est alors calculée, en remplaçant l'argument formel par la valeur (ici 3.5) fournie lors de l'appel. Pour les autres variables qui interviennent (ici A, B, C et D), ce sont leurs valeurs au moment de l'appel qui sont prises en compte.

L'argument peut être fourni lui-même sous forme d'une expression quelconque à calculer :

```
100 PRINT FNA(U*A+EXP(B)/C)
```

Les exemples 30 et 40 ci-dessus montrent qu'il est possible d'utiliser, dans la définition d'une fonction, une fonction déjà définie.

En utilisant cette nouvelle possibilité, voici le programme D-1B qui trace la courbe de la fonction de Gauss entre A et B :

```
1 REM  Programme  D-1B
2 REM
10 INPUT "Bornes";A,B : CLS
15 DEF FNF(X)=EXP(-X^2/2)
20 DEF FNG(X)=1+38*(FNF(X)-MINI)/(MAXI-MINI)
30 MAXI=-10^38:MINI=10^38
40 FOR X=A TO B STEP (B-A)/22
50 IF FNF(X)>MAXI THEN MAXI=FNF(X)
60 IF FNF(X)<MINI THEN MINI=FNF(X)
70 NEXT X
80 FOR X=A TO B STEP (B-A)/22
90 N=INT(FNG(X))
100 PRINT SPC(N);"*":NEXT X
110 GOTO 110
```

La dernière ligne évite l'impression de Ready.

L'avantage du programme D-1B sur la version précédente est qu'il est général : les bornes sont fournies par INPUT et pour changer de fonction à tracer, il suffit de retaper les instructions 15 et 20.

De 30 à 70 s'effectuent les calculs du maximum et du minimum de la fonction, de la même manière qu'à l'exercice 5.10.

Pour le maximum, nous pensions à MAX comme nom de variable. Il s'est avéré qu'on ne peut employer ce nom (mot réservé du BASIC Amstrad) et donc nous avons employé MAXI. Voilà le genre de petite surprise qu'on peut avoir!

Nous avons rencontré dans les programmes de la série B qui précèdent, une bonne occasion d'utiliser un DEF FN; c'est pour imprimer un résultat en ne conservant que deux décimales : la même expression revient plusieurs fois.

Par exemple, dans B-8, on aurait pu écrire :

```
3 DEF FNZ(X)=INT(X*100)/100
```

les lignes 60 et 90 devenant respectivement :

```
60 PRINT "Erreur";FNZ(E);"%" :GOTO 30
90 PRINT "Gagne en";FNZ(SC(J));"secondes"
```

Exercice 7.2. – Reprenez le programme de l'exercice 5.4. En principe, vous l'avez sauvé sur cassette. Vous avez une population répartie en 10 classes. Tracez l'histogramme correspondant.

Un histogramme est un diagramme formé d'autant de bâtons que de classes, chaque bâton ayant une longueur proportionnelle à l'effectif de la classe correspondante. A chaque ligne, ce n'est pas N espaces qu'il faut imprimer, mais N étoiles.

TRACÉS AVEC LOCATE

LOCATE permet d'imprimer où l'on veut sur l'écran. Cela va nous permettre de remettre l'axe des X en horizontal et aussi de tracer des courbes paramétriques (c'est-à-dire définies par x=...; y=..., ce qui est le cas du cercle).

Voici notre programme de tracé de l'exponentielle avec cette méthode.

```
1 REM  Programme  D-1C
2 REM
10 CLS
20 FOR XX=0 TO 1 STEP 1/38
30 X=1+INT(38*XX)
40 Y=22-INT(22*(EXP(XX)-1)/(EXP(1)-1))
50 LOCATE X,Y:PRINT "*"
60 NEXT
```

Exercice 7.3. – Tracer un cercle sur l'écran. *Rappel :* si XC, YC sont les coordonnées du centre et R le rayon, le cercle a pour équations paramétriques :

$$\begin{cases} X=XC+R\cos T \\ Y=YC+R\sin T \end{cases} \qquad 0<=T<=2*PI$$

LE PROBLÈME DE LA RÉSOLUTION

Tous les graphiques que nous avons obtenus jusqu'à maintenant sont assez, ou plutôt, très grossiers puisqu'ils sont définis dans les mailles d'impression de caractères d'où une résolution de 25x40 ou 25x80 au maximum.

En fait, l'Amstrad possède des instructions graphiques permettant une résolution bien meilleure.

Ces instructions graphiques sont utilisables dans chacun des trois modes d'affichage.

En mode 2 on a la résolution maximale : 640 sur 200 suivant le schéma :

Question : Vous dites que la résolution est de 200 en vertical, mais d'après le schéma ci-dessus, il y a 400 valeurs possibles pour y.

– Vous avez raison de poser cette question, car il y a là un point qui doit être parfaitement clair. Il y a 400 valeurs possibles pour la coordonnée y mais la résolution verticale n'est que de 200 car les points de coordonnées x,y et x,y+1 où y est pair correspondent en fait au même point sur l'écran (ce que nous avons appelé un "pixel").

Il y a en somme une notion de points virtuels, définis par leurs coordonnées, opposés aux points réels qui existent sur l'écran. C'est un peu la même notion qui joue dans les deux éléments que nous voyons maintenant : le fait que c'est le même système de coordonnées qui est utilisé dans les modes à résolution moindre et le fait qu'on peut référencer des points hors de l'écran.

En mode 1, la résolution est deux fois moindre dans la direction x, c'est-à-dire qu'on a une matrice 320 (en large) sur 200 (en hauteur). Les coordonnées obéissent au même schéma que pour le mode 2 : simplement toutes les coordonnées telles que :

$$2k \leqslant x \leqslant 2k+1 \quad \text{et} \quad 2l \leqslant y \leqslant 2l+1$$

donneront le même pixel sur l'écran.

De même, en mode 0, la résolution horizontale est encore divisée par 2, la matrice devenant 160 sur 200. Le système de coordonnées reste le même, c'est-à-dire que toutes les coordonnées telles que :

$$4k \leqslant x \leqslant 4k+3 \quad \text{et} \quad 2l \leqslant y \leqslant 2l+1$$

donneront le même pixel. Cela permet d'avoir exactement les mêmes programmes de tracé dans tous les modes : les tracés seront seulement plus grossiers en modes 0 et 1, mais, en mode 2, un pixel ne peut avoir que 2 couleurs : la couleur du fond ou celle de tracé. En mode 1, on a le choix entre 4 couleurs, et en mode 0, on a le choix entre 16 couleurs ; ceci compense cela et, pour notre part, le mode 1 nous semble un excellent compromis.

Voyons maintenant le "tracé" de points hors de l'écran. On peut spécifier des coordonnées telles que le point soit hors de l'écran, par exemple $x>640$ ou $y>400$. Bien sûr le point n'apparaîtra pas, mais cela évite de tester à chaque fois les valeurs des coordonnées. Si l'on spécifie une droite dont l'extrémité est hors de l'écran, seule la portion comprise dans l'écran sera visible : tout se passe comme si l'écran était une fenêtre dans un espace plus vaste.

COMMANDES HAUTE RÉSOLUTION

Il ne nous reste plus qu'à voir les instructions de tracé en haute résolution. Dans chaque mode (fixé par MODE), il y a coexistence d'un écran texte avec son curseur texte, et d'un écran graphique avec son curseur graphique. Le curseur graphique est invisible. Au départ, il est en 0,0 puis il a ensuite la même position que le dernier point tracé.

CLG

Il y a une instruction pour vider l'écran graphique : CLG mais, en fait CLG et CLS vident les deux écrans. La différence est que CLS ramène le curseur texte à l'origine texte (en haut à gauche) et laisse le curseur graphique en place tandis que CLG agit symétriquement : elle laisse le curseur texte en place mais ramène le curseur graphique à l'origine.

ORIGIN

L'origine des graphiques est le point de coordonnées 0,0 situé en bas à gauche de l'écran. On peut changer cette origine par l'instruction ORIGIN X,Y qui place l'origine au point de coordonnées X,Y. Toutes les coordonnées seront alors décomptées par rapport à ce point. La seule exception est l'instruction ORIGIN elle-même dans laquelle les coordonnées sont toujours fournies par rapport au coin en bas à gauche.

Exemple

Si après avoir fait ORIGIN 100,100 vous voulez revenir à l'origine standard, il faut faire ORIGIN 0,0 et non pas ORIGIN −100,−100.

Couleurs

L'indépendance entre graphiques et texte se manifeste aussi à propos des couleurs. Les instructions de tracé ont un paramètre qui détermine la couleur de tracé. Si celui-ci n'est jamais utilisé, les graphiques sont tracés dans la couleur de texte (définie par la dernière instruction PEN). Si celui-ci est utilisé, les graphiques seront tracés dans la couleur spécifiée, tandis que les textes obéiront à PEN. La couleur de graphiques restera valable pour les prochaines instructions graphiques qui ne spécifient pas de couleur. Elle ne changera que lorsqu'une nouvelle instruction graphique spécifiera une nouvelle couleur graphique. Si une instruction PEN est exécutée entre temps, la couleur de texte changera, mais pas la couleur de tracé graphique.

Le 664 possède des instructions GRAPHICS PAPER et GRAPHICS PEN qui permettent d'agir directement sur la couleur des graphiques. Nous ne les utiliserons pas pour rester compatibles avec le 464.

Action sur des points isolés

– **PLOT** : trace un point.

– *Forme la plus souvent utilisée :*
PLOT x,y trace le point de coordonnées x,y dans la couleur courante. Le curseur graphique devient le point de coordonnées x,y.

– *On peut aussi utiliser (surtout en mode 0) :*
PLOT x,y,n trace le point spécifié dans la couleur définie par le dernier INK n,... Cette couleur devient la nouvelle couleur de tracé graphique.

– **PLOTR** : trace un point en coordonnées relatives.

PLOTR dx,dy trace le point qui est défini par les déplacements dx,dy par rapport au curseur graphique. La forme PLOTR dx,dy,n existe aussi.

– **MOVE ou MOVER** : déplace le curseur graphique. De la forme :

MOVE x,y (coordonnées absolues)
MOVER dx,dy (coordonnées relatives)

MOVE et MOVER offrent une manière de positionner le curseur graphique en un point sans l'allumer. On peut aussi spécifier un numéro de stylo, mais il ne sera utilisé que lors du prochain tracé.

Pour éteindre un point, il suffit de faire PLOT (ou PLOTR) en spécifiant comme couleur la couleur du fond.

– **TEST** : cette fonction dit si un point est allumé ou non.

TEST (x,y) : numéro de la couleur du point de coordonnées x,y donc couleur du fond s'il est éteint, couleur de tracé s'il est allumé. Si le point est hors de l'écran, la valeur retournée est 0.

En coordonnées relatives, on emploie la forme :

TESTR (dx,dy).

Pour éteindre un point, mais garder la couleur de tracé qu'on avait pour les tracés subséquents, faire :

```
N=TEST(X,Y):PLOT X,Y,0:MOVE X,Y,N.
```

– **XPOS,YPOS** : ces fonctions donnent respectivement l'abscisse et l'ordonnée du curseur graphique.

Tracés de lignes

– **DRAW ou DRAWR** : permet de dessiner un segment de droite.

– *Forme la plus souvent utilisée :*

DRAW x,y dessine le segment qui va du curseur graphique au point de coordonnées spécifiées. Ce point devient le nouveau curseur graphique.

Avec la forme DRAWR dx,dy les coordonnées sont spécifiées en valeur relative au curseur graphique.

On peut spécifier la couleur du tracé dans DRAW comme dans DRAWR :

DRAW 100,100,1 (tracé en rouge si on a exécuté un INK 1,6 auparavant).

Pour effacer une ligne, il suffit alors de spécifier la couleur du fond.

Le 664 dispose d'une instruction MASK qui permet de tracer des lignes pointillées du motif voulu. Avec le 464, il faut procéder par PLOT.

Tracés globaux

Une seule instruction dans ce groupe et elle n'existe que sur 664.

– **FILL** : permet de colorier une région fermée.

FILL n suppose que le curseur graphique est à l'intérieur d'un certain contour fermé. Alors l'intérieur va être "peint"dans la couleur de numéro n.

Les coordonnées du curseur graphique de départ sont quelconques pourvu qu'elles soient à l'intérieur du contour. Le plus simple est de faire un MOVE juste avant le FILL. Il faut que le contour soit fermé, sinon on risque de peindre tout l'écran. Attention, l'instruction est longue.

Cette instruction est très délicate à simuler avec le 464 (qui ne l'a pas, hélas !). Il faut faire des DRAW de lignes rapprochées. C'est très long.

Nous sommes maintenant prêts pour des exemples et des exercices.

Il y a toutefois encore un point à noter : lorsqu'un programme vient de faire un dessin en mode graphique et qu'il se termine, il affiche Ready en mode texte. Si vous ne voulez pas que cela trouble votre dessin, il suffit que la dernière instruction soit une attente, par exemple :

```
100 X$=INKEY$ : IF X$ =""GOTO 100
```

qui attend la frappe d'un caractère quelconque au clavier.

Nous sommes maintenant en mesure de tracer une courbe en haute résolution.

Nous allons tracer quatre alternances d'une sinusoïde. Nous adopterons l'équation : Y=200+190*SIN(8*PI*X/640) d'où le programme :

```
1 REM  Programme D-2A
2 REM
10 MODE 1:INK 0,0:INK 1,26
20 FOR X=0 TO 639
30 Y=200+190*SIN(8*PI*X/640)
40 PLOT X,Y
50 NEXT
60 X$=INKEY$:IF X$="" GOTO 60
```

Ci-dessous, photo du résultat (en mode 1). Nous vous engageons vivement à modifier l'instruction 10 pour observer la différence avec les modes 0 et 2.

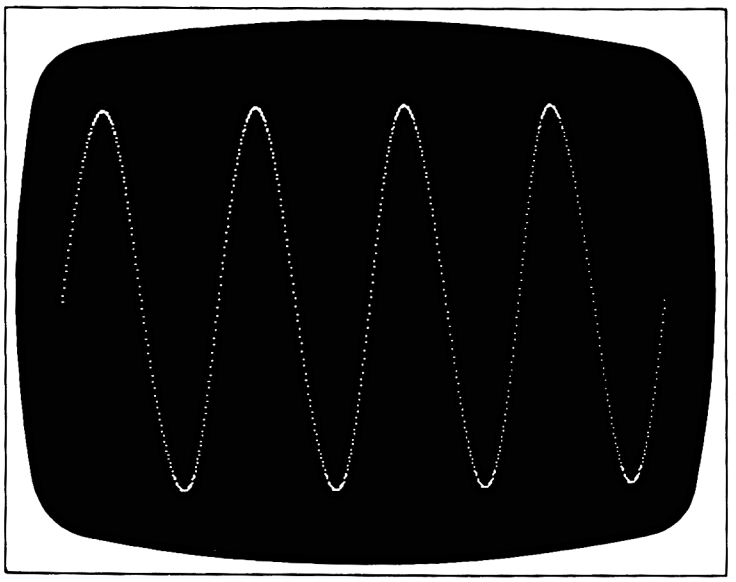


Photo 3

Exercice 7.4. – La courbe est un peu trop pointillée. En fait, ce qu'il faut, c'est tracer la courbe sous forme de petits segments.

MÉLANGE TEXTE-DESSIN

On peut parfaitement comme on l'a vu, écrire du texte sur un écran graphique. Mais le texte s'écrira en couleur texte et à la position du curseur texte. Pour écrire à l'emplacement du curseur graphique, il faut employer TAG puis

des PRINT terminés par ; sinon les caractères de contrôle seront affichés et l'on termine par TAGOFF. A part le fait que ceci autorise un positionnement plus fin, on peut se borner à utiliser LOCATE et PRINT sur l'écran texte.

Le programme D-2C produit l'affichage montré sur la photo 4. On est parti du programme de tracé continu de l'exercice 7.4. en changeant un peu les paramètres pour laisser la place du titre.

```
1 REM  Programme D-2C
2 REM
10 MODE 1:INK 0,0:INK 1,26:PLOT 0,200
20 FOR X=0 TO 639
30 Y=200+180*SIN(8*PI*X/640)
40 DRAW X,Y
50 NEXT
60 LOCATE 15,25 :PRINT "SINUSOIDE";
70 X$=INKEY$:IF X$="" GOTO 70
```

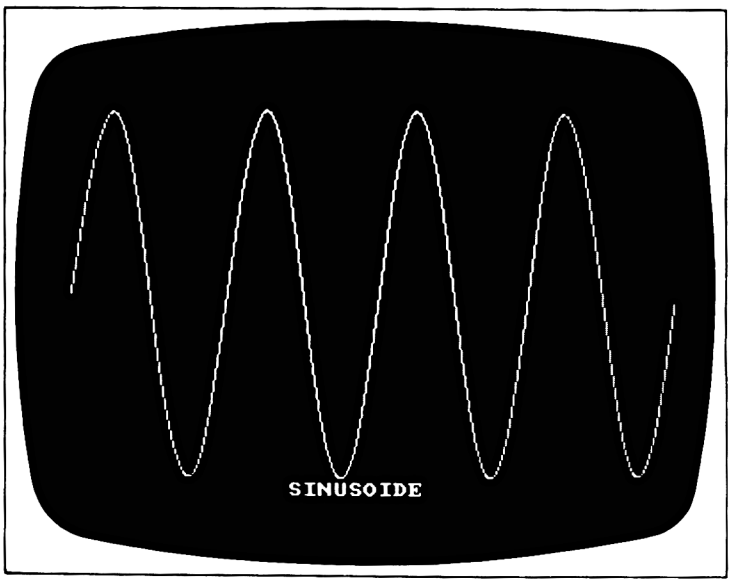


Photo 4

Le programme D-2D est le même, mais en MODE 0. On voit (cf. *photo 5*) combien le tracé est plus grossier et aussi que les lettres affichées sont très grosses (cela peut avoir des applications). C'est pourquoi on a changé un peu les paramètres de la courbe pour laisser la place.

```
1 REM  Programme D-2D
2 REM
10 MODE 0:INK 0,0:INK 1,26:PLOT 0,200
20 FOR X=0 TO 639
30 Y=200+170*SIN(8*PI*X/640)
40 DRAW X,Y
50 NEXT
60 LOCATE 7,25 :PRINT "SINUSOIDE";
70 X$=INKEY$:IF X$="" GOTO 70
```

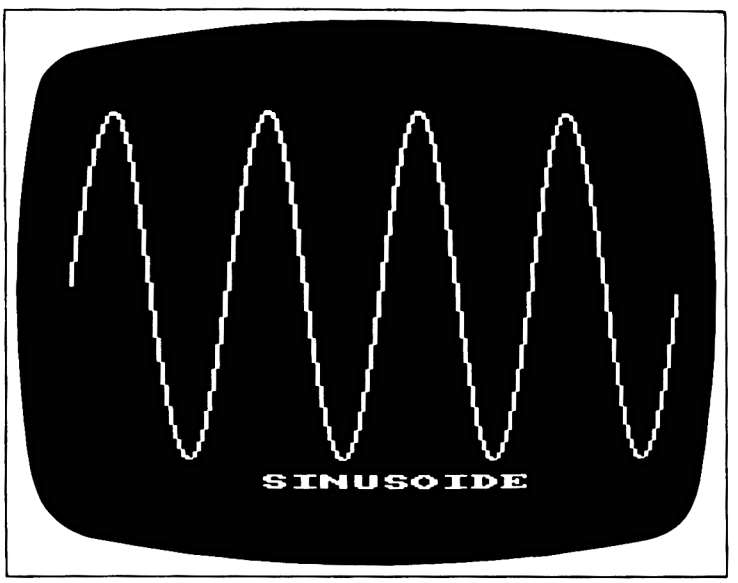


Photo 5

Nous vous conseillons aussi d'essayer en mode 2.

Exercice 7.5. – Dessinez un cercle, un disque orangé et enfin un disque clignotant.

Exercice 7.6. – Faire un pavage de couleurs aléatoires en mode 0.

Exercice 7.7. – Transformez votre Amstrad en Télécran. En fonction de la touche enfoncée, nous déplaçons le point de tracé. Si la touche SHIFT n'est pas enfoncée, il y a simple déplacement. Si elle est enfoncée, il y a tracé. ASC(INKEY$+CHR$(0)) est à tout moment le code de la touche actuellement enfoncée.

Nous proposons les affectations suivantes :

Touche	Code	Mouvement	Touche	Code	Mouvement
aucune curseur[1]	0		COPY	224	aller au centre de l'écran
→	243	à droite	espace	32	en bas à droite
↑	240	en haut	DEL	127	retour en haut à gauche
←	242	à gauche			
↓	241	en bas	CLR	16	vidage d'écran

(1) On ajoute 4 lorsque SHIFT est enfoncé. *Ex.* on a 247 si l'on fait SHIFT curseur droite.

Maintenant que nous avons des affichages très élaborés, il nous reste à donner un peu de mouvement.

NOTIONS SUR LES DESSINS ANIMÉS

Nous nous bornerons ici aux premières notions permettant d'animer l'affichage de l'Amstrad.

La première méthode qui vient à l'esprit est la méthode classique utilisée au cinéma. La scène est décomposée en un grand nombre de dessins qui sont projetés successivement.

Avec l'Amstrad, qui est pourtant un des micro-ordinateurs personnels les plus rapides, se pose le problème du temps de remplissage de l'écran pour obtenir une image. D'autant plus que, pour ne pas avoir trop de scintillement, il faut afficher au moins 15 images par seconde. Ceci nécessite le recours au langage machine.

Pour pouvoir utiliser BASIC, il faut passer d'une image à la suivante en ne changeant qu'une très petite partie de l'image.

Mouvements d'une balle

A titre d'exemple d'animation sans modification importante d'une image à la suivante, nous allons faire se déplacer une balle sur l'écran. Cela conduira à un embryon de jeu de tennis. La balle est le caractère de code 231. On peut aussi prendre "*".

Pour déplacer la balle de gauche à droite, il faut :

- imprimer la balle

```
20 X=... :Y=... :LOCATE X,Y :PRINT "*"
```

– revenir sur la balle et la remplacer par un blanc, puis imprimer une nouvelle balle un cran plus loin.

```
30 LOCATE X,Y :PRINT " "  :X=X+1 :LOCATE X,Y :PRINT"*"
```

– boucler sur l'instruction 30

```
40 GOTO 30
```

Essayez le programme formé par les trois instructions 20, 30 et 40 ci-dessus.

Il y a plusieurs défauts :

– il faudrait vider l'écran et se placer sur la ligne centrale (13);

– lorsque la balle a parcouru tout l'écran de gauche à droite, on a un message d'erreur;

– il n'y a pas de règlage de vitesse.

La première objection se résout facilement par le choix de Y=13.

Pour la troisième, il faut introduire un délai avant de faire le GOTO en 40, par exemple à l'aide d'une boucle :

```
40 FOR T=1 TO D : NEXT : GOTO 30
```

D sert à règler le délai, donc la vitesse de déplacement.

Pour la deuxième objection, on peut décider le mouvement suivant : aller de gauche à droite, puis, quand la balle arrive à l'extrémité de la ligne, aller de droite à gauche, etc.

Pour cela, il faut tester X. Le sens est inversé quand X = 40 ou 1.

Pour que la balle se déplace à gauche, il faut écrire :

```
50 LOCATE X,Y :PRINT " "  :X=X-1 :LOCATE X,Y :PRINT"*"
```

D'où le programme :

```
1 REM  Programme  D-4
2 REM
10 CLS
20 X=1:Y=13:D=40:LOCATE X,Y:PRINT "*"
30 LOCATE X,Y:PRINT " ":X=X+1:LOCATE X,Y:PRINT "
*"
40 FOR T=1 TO D:NEXT:IF X<40 GOTO 30
50 LOCATE X,Y:PRINT " ":X=X-1:LOCATE X,Y:PRINT "
*"
60 FOR T=1 TO D:NEXT:IF X>1 GOTO 50
70 GOTO 30
```

Avec D=50, vous réalisez un véritable hypnotiseur. Avec D<10, la balle va assez vite pour que l'on croie voir un sillage. Avec D>80, les saccades du mouvement sont apparentes.

Exercice 7.8. – Faire osciller la valeur du délai entre 1 et 80. (Le délai varie de 1 à chaque aller et retour.)

Exercice 7.9. – Faire osciller la balle du haut en bas de l'écran.

Exercice 7.10. – Faire aller la balle en diagonale (du coin supérieur gauche à la dernière ligne position 25 et retour).

Tennis

Nous sommes prêts maintenant à jouer au tennis. En fait, notre jeu sera rudimentaire. Partons de la balle qui oscille de gauche à droite, nous disposons un joueur à droite et un joueur à gauche. Le joueur de droite ne doit pas laisser la balle passer la position 39 sur la ligne. Pour manifester qu'il rattrape la balle, il doit appuyer sur une touche. Nous adopterons la touche \. Pour le joueur de gauche, ce sera la touche Z et la position 2. Le programme reconnaîtra la touche grâce à un INKEY$.

Par ailleurs, nous imposons aux joueurs de ne pas appuyer trop tôt sur leur touche : si le joueur de droite appuie sur \ avant que la balle ne soit en colonne 36, ou si le joueur de gauche appuie sur Z avant qu'elle ne soit en colonne 5, il concède un point à son adversaire.

A chaque point, le score est affiché. Le service sera effectué en appuyant sur une touche quelconque. Lors du service, la balle part du milieu de l'écran. Le sens du mouvement est tiré au sort.

Enfin on a implanté le mouvement de la balle dans un sous-programme : 400.

D'où le programme D-5 dont il y aurait lieu de perfectionner la présentation pour obtenir un jeu effectif.

```
1 REM  Programme D-5
2 REM   MINI-TENNIS
3 REM
5 SG=0:SD=0:D=20:X=1
10 CLS
15 PRINT CHR$(30);:PRINT SG;SPC(32);SD:K=20:A$=I
NKEY$:IF A$="" GOTO 15
20 IF RND(1)>0.5 GOTO 45
25 FOR T=1 TO D:NEXT:K=K+1:IF K=40 THEN SG=SG+1:
GOTO 15
30 GOSUB 400:A$=INKEY$:IF A$<>"ç" GOTO 25
35 IF K>=36 GOTO 45
40 SG=SG+1:GOTO 10
45 FOR T=1 TO D:NEXT:K=K-1:IF K=1 THEN SD=SD+1:G
OTO 15
50 GOSUB 400:A$=INKEY$:IF A$<>"z" GOTO 45
55 IF K<=4 GOTO 25
60 SD=SD+1:GOTO 10
400 LOCATE X,13:PRINT " ":X=K:LOCATE X,13:PRINT
"*":RETURN
```

Il nous reste à voir trois questions un peu spécialisées. Vous pouvez les éviter en première lecture.

Les fenêtres

On peut définir au plus huit fenêtres d'affichage, de numéros allant de 0 à 7. Une fenêtre se définit par :

```
WINDOW #n,gauche,droite,haut,bas
```

où n est le numéro de fenêtre (de 0 à 7) et les autres paramètres ont une signification évidente. Selon le mode, droite doit être inférieur à 20, 40 ou 80 ; bas a pour limite 25.

Le principe même de la fenêtre est qu'elle forme un sous-espace vis-à-vis du déroulement sur l'écran, c'est à dire que si des écritures dans une fenêtre entraînent un déroulement, il n'y a pas de mouvement dans une autre fenêtre qui entourerait la fenêtre considérée.

On peut aussi définir une fenêtre graphique, mais c'est obligatoirement la fenêtre numéro 0. Cela se fait par une forme plus complète de l'instruction ORIGIN :

```
ORIGIN x,y,gauche,droite,haut,bas
```

mais les paramètres de dimension sont, cette fois, fournis en unités de coordonnées graphiques. Il n'y a pas de numéro puisque c'est toujours 0.

Les instructions CLS, LOCATE, PAPER, PEN et PRINT se généralisent aux fenêtres sous la forme :

```
CLS #n
LOCATE #n,x,y
PAPER #n,encre
PEN #n,encre
PRINT #n,liste
```

On peut aussi lister dans une fenêtre par LIST lignes,#n (remarquer la dissymétrie de la syntaxe !) et lire au clavier dans une fenêtre par INPUT #n,"texte";liste.

Exemple

Pour voir apparaître simultanément sur l'écran un programme et un sous-programme, on peut faire :

```
MODE 2
WINDOW #0,1,38,1,25 :WINDOW #1,41,80,1,25
LIST -500 :LIST 1000-,#1
```

Plusieurs fenêtres peuvent se recouvrir en tout ou partie, mais, sauf si l'on emploie des modes transparents, ce qu'on écrit en dernier dans une fenêtre, empêche de voir ce qu'il y avait dans une fenêtre recouverte.

L'instruction WINDOW SWAP n1,n2 échange les rôles des fenêtres n1 et n2 : ce qui devait s'inscrire en n1 ira en n2 et inversement. Par exemple, après WINDOW SWAP 0,1 les messages d'erreur de BASIC iront dans la fenêtre 1.

Les modes de superposition

Que se passe-t-il lorsque vous tracez un trait jaune et que celui-ci traverse un trait rouge tracé auparavant ? Eh bien, le point de croisement devient jaune. C'est le mode le plus normal, qu'on appelle mode "forcé" et que vous avez si vous ne faites rien. Mais vous pouvez avoir d'autre modes.

En fait (cf. section suivante) faire un tracé en un point revient à écrire un motif binaire en mémoire. En mode forcé, on écrit tout simplement le nouveau motif binaire voulu. Dans les autres modes, on combine l'ancien et le nouveau motifs par un OR, un AND ou un XOR. En mode 2, où les 1 du motif binaire signifient "allumé" et les 0 signifient "éteint", les modes forcé, AND et OR allument les points de croisement. En mode 0 ou 1, les modes de superposition entraînent divers effets de combinaison de couleurs dans les zones de croisement ; leur étude dépasse le cadre de ce livre.

On choisit le mode en faisant imprimer le caractère de contrôle de code 23 suivi d'un code 0, 1, 2 ou 3 pour imposer respectivement "forcé", XOR, AND ou OR.

Dans le même ordre d'idées, on peut choisir entre les modes d'écriture "transparente" et "opaque". En mode opaque, la dernière écriture cache celles qui précèdent. En mode transparent, on peut voir à travers (c'est le fond qui est transparent, d'où l'intérêt pour les fenêtres). Sur 664, le choix peut se faire dans les instructions PEN et GRAPHICS PEN. Sur les deux modèles, on peut procéder par impression du caractère de contrôle de code 22 suivi d'un code 0 (opaque) ou 1 (transparent).

Inscription sur l'écran par POKE

Il existe une autre manière d'écrire ou de dessiner sur l'écran. Cette manière dérive des deux possibilités d'accès direct à la mémoire offertes par l'Amstrad, les instructions PEEK et POKE.

PEEK permet de lire où l'on veut en mémoire.

PEEK(X) fournit la valeur comprise entre 0 et 255 de l'octet d'adresse X (en décimal ou en hexa).

Par exemple, ?PEEK(0) fait imprimer 1 car, à l'adresse 0 de la mémoire, il y a 1.

Si la première instruction de votre programme est 5, alors ?PEEK(370) fait imprimer 5. Vous pouvez ainsi prendre connaissance du contenu de tout emplacement mémoire.

POKE permet d'écrire à un emplacement mémoire.

POKE X,Y écrit à l'adresse X la valeur Y (Y doit être compris entre 0 et 255, X entre 0 et 65535).

Vous pouvez présenter les paramètres de PEEK et POKE sous forme hexadécimale (&Hxxx) ou binaire.

Par exemple, faites POKE 370,6 : la première instruction de votre programme aura maintenant le numéro 6. On conçoit que PEEK et POKE permettent des effets très astucieux, mais qui dépassent le cadre de ce livre.

Un de ces effets est l'écriture directe en mémoire d'écran. Il y a en effet une zone mémoire consacrée à l'écran, d'adresses 49152 (&HC000) à 65151 suivant le schéma :

```
49152 49153 49154          49230 49231
49232 49233 49234          49310 49311
    :                          :
    :                          :
    :                          :
    :                          :
65072 65073                65150 65151
```

Les 16000 octects sont répartis en 200 lignes élémentaires de 80 octets. Quelque soit le mode (0, 1 ou 2) et que l'on écrive du texte ou du dessin, on a la même répartition. Cela signifie en particulier que, si l'on veut écrire du texte, il faut dessiner les lettres point par point, ce qui est très fastidieux.

Un octet représente toujours la même zone sur l'écran, qu'on soit en mode 0, 1 ou 2. En mode 2, l'octet représente 8 pixels fins. Il y a donc 1 bit par pixel, 0=encre 0, 1=encre 1.

En mode 1, l'octet représente 4 pixels deux fois plus gros. Mais on a 2 bits par pixel, ce qui permet 4 couleurs. La répartition est un peu compliquée, elle obéit au schéma :

p1bit1 p2bit1 p3bit1 p4bit1 p1bit0 p2bit0 p3bit0 p4bit0

où les pixels vont de gauche à droite de p1 à p4. Le numéro de couleur d'un pixel est formé des 2 bits 1 et 0 (le motif binaire est pris numéro de bit croissant de gauche à droite).

En mode 0, l'octet représente 2 gros pixels, avec 4 bits de couleur pour chacun, ce qui permet 16 couleurs. On a le schéma :

p1bit3 p2bit3 p1bit2 p2bit2 p1bit1 p2bit1 p1bit0 p2bit0

Par exemple, en mode 1, POKE 49152,&X10001111 dessine un petit segment dont le pixel le plus à gauche est rouge et les trois autres bleu clair. De toutes façons l'utilisation de ceci est compliquée, donc peu recommandée et les instructions BASIC "évoluées" dont le BASIC de l'Amstrad est particulièrement riche devraient permettre de tout faire sans recours à PEEK et POKE.

Récapitulation

Nous avons maintenant vu la plupart des techniques de production d'images accessibles à l'Amstrad. Certaines sont très perfectionnées comme l'affichage haute résolution.

A propos des dessins animés, nous avons vu que le BASIC est un peu lent pour certaines applications.

Dans ce cas, il y a un recours; c'est le langage machine du microprocesseur de l'Amstrad.

Nous allons maintenant rendre notre Amstrad sonore.

CHAPITRE 8
EFFETS SONORES

Voilà un chapitre qui va faire du bruit. En effet, l'Amstrad est capable de générer des sons variés sur le haut-parleur qui lui est incorporé. Le volume est commandé par le bouton situé à droite de l'Amstrad. Il y a en outre une sortie (stéréo) qui permet de diriger les sons vers votre chaîne HiFi.

Nous citons pour mémoire le fait que PRINT CHR$(7) ou PRINT "CTRL G" produit un bip.

L'instruction principale de production d'un son sur l'Amstrad est SOUND. Comme les paramètres sont nombreux et ont des options par défaut, elle revêt des formes diverses. Voici la forme minimale :

SOUND voix,période

où voix détermine le numéro de l'une des trois voies (ou voix ! ou canaux) que l'on veut faire retentir et période est un nombre compris entre 1 et 4095 proportionnel à la période de la note à jouer. La période est l'inverse de la fréquence donc plus la note est haute, plus la valeur est petite. On peut spécifier 0 pour les effets de bruits.

Les valeurs des notes sont précisées par le tableau suivant. Par exemple SOUND 1,142 fait retentir un la.

Tableau 8.1. – Valeurs des notes

Note	Valeur	Note	Valeur	Note	Valeur	Note	Valeur
do −1	3822	do 0	1911	do 1	956	do 2	478
do#	3608	do#	1804	do#	902	do#	451
ré	3405	ré	1703	ré	851	ré	426
ré#	3214	ré#	1607	ré#	804	ré#	402
mi	3034	mi	1517	mi	758	mi	379
fa	2863	fa	1432	fa	716	fa	358
fa#	2703	fa#	1351	fa#	676	fa#	338
sol	2551	sol	1276	sol	638	sol	319
sol#	2408	sol#	1204	sol#	602	sol#	301
la	2273	la	1136	la	568	la	284
la#	2145	la#	1073	la#	536	la#	268
si	2025	si	1012	si	506	si	253
do 3	239	do 4	119	do 5	60	do 6	30
do#	225	do#	113	do#	56	do#	28
ré	213	ré	106	ré	53	ré	27
ré#	201	ré#	100	ré#	50	ré#	25
mi	190	mi	95	mi	47	mi	24
fa	179	fa	89	fa	45	fa	22
fa#	169	fa#	84	fa#	42	fa#	21
sol	159	sol	80	sol	40	sol	20
sol#	150	sol#	75	sol#	38	sol#	19
la	142	la	71	la	36	la	18
la#	134	la#	67	la#	34	la#	17
si	127	si	63	si	32	si	16

On a indiqué les numéros des octaves. Le do du milieu du piano et le la du diapason sont dans l'octave 3. Bien sûr, les valeurs sont approximatives surtout dans l'octave la plus aiguë.

Une forme plus complète est :

SOUND voix,hauteur,durée,volume

où durée indique la durée de la note en centièmes de seconde (valeur par défaut 20 soit 0,2 s). Une valeur négative (−n) signifie "répéter l'enveloppe n fois". Volume (de 0 - silence - à 15 le plus fort -) définit le volume du son. La valeur par défaut est 12.

La forme complète est :

SOUND voix,hauteur,durée,volume,enveloppe,modulation,bruit

où enveloppe est un numéro d'enveloppe qui définira la variation d'amplitude du son au cours de son déroulement. Une enveloppe est définie par une instruction ENV. Modulation est aussi un numéro d'enveloppe, mais faisant varier la hauteur du son pour produire, par exemple, du vibrato. Une modulation est définie par une instruction ENT. SOUND fait référence à un numéro d'enveloppe ou de modulation par autant d'instructions ENV et ENT (au plus 15 de chaque). Bruit est une période qui permet d'ajouter des effets de bruit. Si hauteur<>0 et bruit=0, on a un son pur ; si hauteur=0 et bruit<>0, on a un bruit pur et si les deux sont nuls, on a un mélange son-bruit.

LES FILES D'ATTENTE

Une propriété essentielle de l'instruction SOUND est qu'elle lance le son et redonne immédiatement le contrôle à l'utilisateur. Cela permet de lancer une autre note sur une autre voix et, ainsi, de produire un accord.

Bien entendu, si vous envoyez une deuxième instruction SOUND sur la même voix, le son attendra que le premier son soit terminé pour démarrer. Mais le contrôle vous sera tout de même rendu : les sons envoyés à une voix sont mis en file d'attente : il peut y avoir jusqu'à 5 sons en attente sur une voix et, bien sûr, si la file d'attente est pleine, une nouvelle instruction SOUND sur la voix considérée met le programme en attente.

Deux autres phénomènes d'attente peuvent intervenir. Lorsqu'on spécifie un son sur un canal (voix), on peut spécifier un "rendez-vous" avec un autre canal, c'est-à-dire qu'il démarre lorsque le prochain son du canal de rendez-vous démarrera. C'est ce qui permet des accords bien simultanés.

Enfin, on peut spécifier que le son considéré soit en attente et qu'il ne démarre que lorsque l'on exécutera une instruction RELEASE voix sur la combinaison de canaux indiquée.

LE PARAMÈTRE "VOIX"

Il nous faut voir maintenant comment l'on spécifie le canal. Le paramètre "voix" spécifie en fait une combinaison de canaux. Le paramètre "voix" est

un octet dont chaque bit a une signification précise obéissant au schéma suivant :

Tableau 8.2. – Le paramètre "voix"

Bit	Valeur	Fonction
0	1	jouer sur voix 1
1	2	jouer sur voix 2
2	4	jouer sur voix 3
3	8	rendez-vous avec voix 1
4	16	rendez-vous avec voix 2
5	32	rendez-vous avec voix 3
6	64	mise en attente
7	128	vidage de la file d'attente de la voix

Les valeurs de la deuxième colonne sont intéressantes. Pour activer un certain ensemble de fonctions, il suffit de spécifier la somme des valeurs correspondantes.

Exemple

– Pour envoyer la même note sur les canaux 1 et 3, comme 1+4=5, il suffit de faire SOUND 5, ...

– Pour envoyer une note sur la voix 1 avec rendez-vous avec la voix 2, faire SOUND 17, ...

– Note sur la voix 2, avec attente : SOUND 66, ...

– Libération des attentes sur les trois canaux (ce qui permet de démarrer un accord) : RELEASE 7.

La fonction 128 vide la(les) file(s) d'attente de la(des) voix spécifiée(s) et donc SOUND 135, ... arrête tous les sons. On peut le faire aussi par PRINT "CTRL G".

Bien sûr, les paramètres peuvent être fournis sous forme d'expressions arithmétiques ou de valeurs lues en DATA ou de toute autre manière favorable.

Lorsque l'on spécifie deux notes successives identiques sur une voix, l'Amstrad les enchaîne de sorte qu'elles n'en forment qu'une seule, de durée égale à la somme des durées :

```
SOUND 1,142,100 : SOUND 1,142,100
```

est équivalent à :

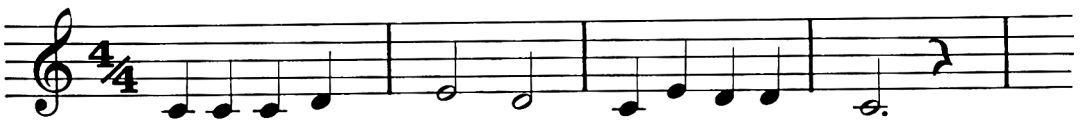
```
SOUND 1,142,200
```

Si l'on veut deux sons séparés, il faut intercaler un court silence, obtenu, par exemple, avec un volume nul :

SOUND 1,142,100 : SOUND 1,142,10,0 : SOUND 1,142,100.

Une autre façon serait d'utiliser une enveloppe se terminant par une petite portion silencieuse.

Exercice 8.1. – Jouer *Au clair de la Lune* (dododorémi,ré,domirérédo).



Exercice 8.2. – Produire l'accord DO MI SOL.

Exercice 8.3. – Dans le jeu du quiz-géographique, quand le jouer a gagné, faites retentir les premières notes de la *Marseillaise* (ré ré, ré sol, sol la, la ré, si sol).

Exercice 8.4. (jeu de Simon - marque déposée) – On établit une correspondance entre les touches 1 à 7, 7 couleurs, et les notes de la gamme. Le programme choisit au hasard une séquence de 4 notes : il la fait retentir tout en affichant sur tout l'écran les couleurs correspondantes. Le joueur doit alors taper la séquence de touches correspondante. Pendant qu'il appuie sur une touche, le programme fait retentir la note et affiche la couleur. On autorise deux tentatives pour trouver une séquence.

LES ENVELOPPES

On a vu comment l'on spécifie l'usage d'une enveloppe dans l'instruction SOUND. L'enveloppe doit préalablement avoir été définie par une instruction ENV de la forme :

ENV enveloppe,sections

où enveloppe est le numéro d'enveloppe tel qu'il est référencé dans une instruction SOUND. Il peut y avoir au plus 5 sections de variation, qui sont chacune de la forme :

nombre de pas,valeur du pas,temps

La valeur du pas indique de combien d'unités de volume on augmente ou l'on diminue. Le temps de chaque pas est en centièmes de secondes. Voici une enveloppe de clavecin :

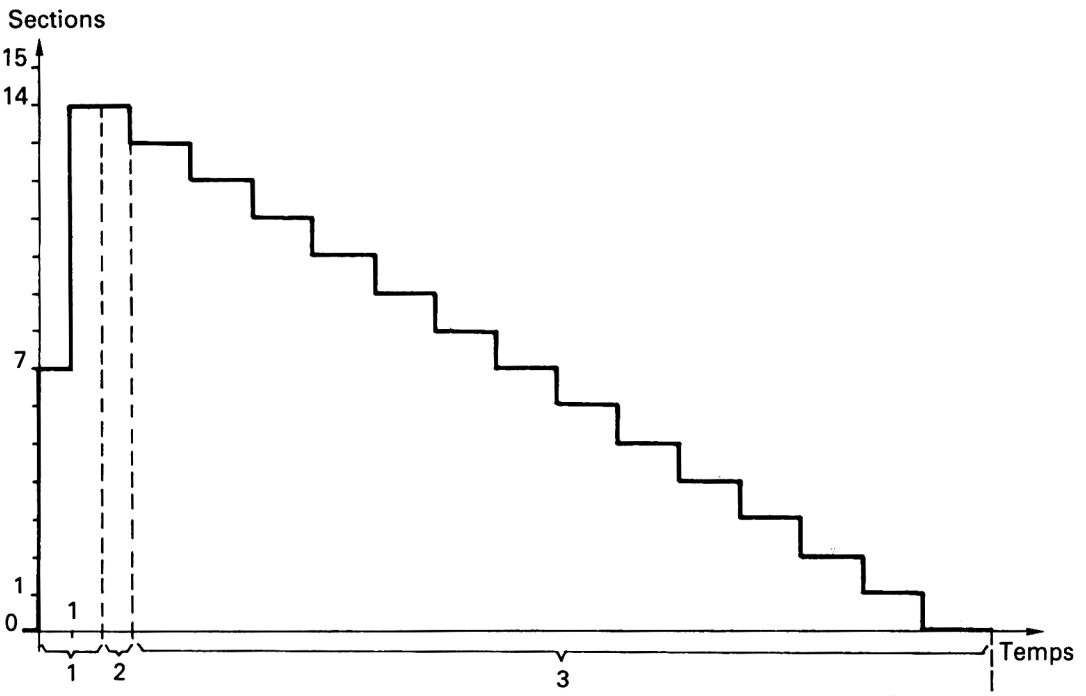


Figure 8.1.

Elle est produite par ENV 1,2,7,1,1,0,1,14,−1,2. Pour l'expérimenter, faites SOUND 1,142,0,0,1. SOUND 1,142,−3,0,1 fait retentir la note 3 fois.

Cette enveloppe a trois sections : une montée rapide, un court palier (1 pas à variation nulle) et une descente plus lente. Si vous changez le dernier 2 en 1, vous avez le mode étouffé du clavecin. Si vous le changez en 4, le son ressemble à celui du piano.

Le volume spécifié dans SOUND sert d'amplitude de départ. Si dans une section, le volume est amené en dessous de 0, il reprend à 15 et s'il est amené au-dessus de 15, il reprend à 0 ; on obtient donc des dents de scie.

La relation avec la durée spécifiée dans SOUND est complexe. De toutes façons, l'enveloppe a une durée bien déterminée, somme des durées des sections. La durée de chaque section est nombre de pas x temps de chaque pas. Si la durée spécifiée est 0, alors la durée du son sera celle de l'enveloppe. Si la durée spécifiée est négative (−n), l'enveloppe sera répétée n fois, donc la durée du son sera n fois la durée de l'enveloppe. Si la durée spécifiée est positive, soit d1, d2 étant la durée de l'enveloppe, il y a deux cas :

1. – d1<d2 : la durée du son sera d1, c'est-à-dire que l'enveloppe sera interrompue en cours de déroulement ;

2. – d1>d2 : la durée du son sera d1, les d2 premiers centièmes de seconde suivant l'enveloppe. Le son se terminera par d1–d2 centièmes de seconde faits avec une amplitude constante égale à la valeur atteinte en fin d'enveloppe.

Exercice 8.5. – Produire l'enveloppe figure 8.2. (essai de simuler une descente exponentielle).

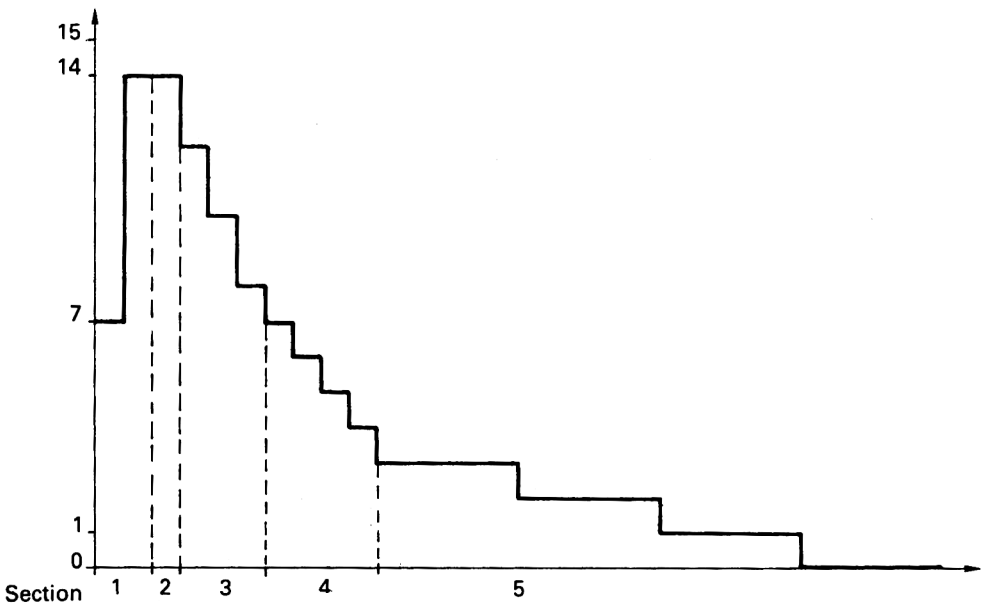


Figure 8.2.

Les modulations sont à la hauteur ce que les enveloppes sont à l'amplitude. D'ailleurs Amstrad les appelle "enveloppes de ton". L'instruction de spécification de modulation ENT a exactement la même forme que ENV :

ENT modulation,sections

où le premier paramètre est le numéro de modulation qui sera référencé dans SOUND. Comme pour ENV, on a au plus 5 sections de la forme : nombre de pas,variation,temps. Le temps est en centièmes de seconde et la variation (positive ou négative) s'exprime en unités de période.

La modulation a, elle aussi, une durée, mais sa relation avec la durée du son est différente de celle de l'enveloppe. Soit d la durée résultant de l'enveloppe et de la durée spécifiée dans SOUND et d3, la durée de la modulation.

Si d3>d, la durée du son sera d et il y aura une partie de la modulation inutilisée.

Si d3<d, la durée du son sera d avec une partie modulée de durée d3 puis d–d3 centièmes de seconde avec hauteur constante égale à la valeur atteinte en fin de modulation.

Il y a un cas particulier lorsque d3<d : si, dans ENT, on a spécifié le numéro de modulation négatif (−n, attention, il doit quand même être spécifié positif : n, dans SOUND), alors la modulation est périodique, c'est-à-dire qu'elle est répétée jusqu'à écoulement de la durée d.

Ce dernier cas est idéal pour le vibrato ; essayez :

```
ENT −1,4,1,1,8,−1,1,4,1,1
SOUND 1,142,1000,15,,1
```

Remarquez les virgules consécutives dans SOUND qui marquent l'absence de spécification d'enveloppe. Naturellement, l'amplitude de variation doit être adaptée aux notes jouées : ici, à partir du la, on va à mi-chemin du sol dièse et du la dièse. En outre, la vitesse de vibrato doit être adaptée au tempo du morceau.

Exercice 8.6. – Reconstituez le schéma du vibrato précédent. Modifiez l'instruction pour qu'il se fasse deux fois plus lentement.

Exercice 8.7. – Faire retentir une sirène.

LES BRUITS

Le dernier paramètre de l'instruction SOUND permet de produire des bruits, c'est-à-dire des mélanges aléatoires de fréquences situées autour d'une fréquence centrale dont le dernier paramètre (qui doit être compris entre 1 et 31) est proportionnel à la période.

Le programme ci-dessous imite un coup de feu.

```
1 REM  Coup de feu
2 REM
10 ENV 1,2,7,1,1,0,1,3,-2,1,4,-1,1,4,-1,4
20 SOUND 1,0,0,0,1,,10
```

On a adopté l'enveloppe "clavecin à décroissance exponentielle".

Nous vous conseillons d'expérimenter : changez l'enveloppe ; changez la fréquence centrale du bruit. On peut tabler sur les valeurs suivantes (elles doivent être comprises entre 1 et 31) :

1	sifflements
2, 3, 4	réacteurs, fuites de gaz
5 - 8	tempête, lance flamme
9 - 16	vent
17 - 22	machines, explosion
23 - 31	avions, grondements

Les enveloppes périodiques peuvent donner de bons effets ; essayez le programme ci-dessous :

```
1 REM  Moteur de rafiot
2 REM
10 ENV 1,1,15,1,3,-5,2,50,0,50
20 SOUND 1,0,10,0,1,,10
30 GOTO 20
```

Si besoin est, pour arrêter le bruit, faites :

PRINT " CTRL G ".

Nous terminons ce chapitre en citant la fonction SQ(voix) qui fournit l'état du canal indiqué : les bits ont les rôles suivants :

bits 0 à 2	nombre de places libres dans la file d'attente
bits 3, 4 ou 5	rendez-vous attendu avec la voix 1, 2 ou 3
bit 6	ce canal est en attente
bit 7	ce canal est en train de jouer.

Par exemple, IF SQ(1)<128 THEN SOUND 1, ... renvoi un son sur la voix 1 dès qu'elle ne joue plus.

Récapitulation

Nous avons maintenant eu un aperçu de la plupart des techniques de programmation accessibles à l'Amstrad tant dans les domaines du calcul que dans celui des jeux, des traitements graphiques et des effets sonores.

CHAPITRE 9
L'AMSTRAD ET LE MONDE EXTÉRIEUR

Nous faisons dans ce chapitre un bref tour de notre Amstrad ou plutôt de ses connecteurs et des périphériques qu'on peut y brancher. Nous donnons ci-dessous le schéma de la disposition des connecteurs.

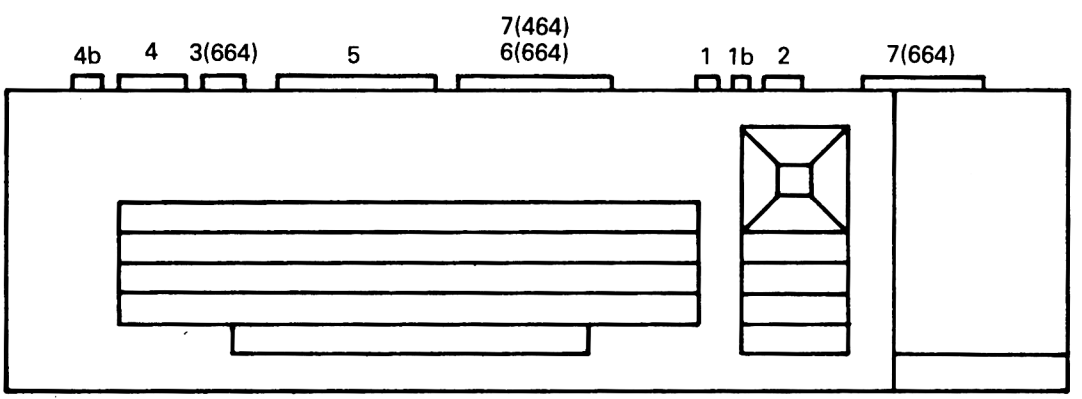


Figure 9.1. – Les connecteurs de l'Amstrad

1. – Le connecteur d'alimentation. Il est relié au moniteur. L'ordinateur et le moniteur ont chacun leur propre bouton marche/arrêt. Les deux boutons (celui de l'ordinateur et celui du boîtier) doivent être en position marche. On peut maintenir en marche celui de l'ordinateur et l'on allume à l'aide de celui du moniteur. Le 664 a un deuxième cordon (1b) pour le 12 volts.

2. – Le connecteur vidéo (6 broches). Il est aussi relié au moniteur.

3. – Le connecteur magnétophone (664 seulement - 5 broches). Il permet d'utiliser un magnétophone ordinaire, ce qui offre la possibilité, sur 664, de récupérer des logiciels de 464. Il est recommandé que votre magnétophone ait une commande à distance.

4. – Il y a un connecteur (9 broches) pour manettes de jeu (joysticks). On peut y connecter des manettes à levier et, pour cela, vous pouvez prendre des

modèles Atari, Commodore ou compatibles. Pour connecter un deuxième joystick, il faut avoir le modèle Amstrad, qui a un deuxième connecteur.

Le BASIC Amstrad possède une fonction qui facilite grandement l'utilisation de ces joysticks :

JOY(X) où X est le numéro 0 (premier joystick) ou 1 (deuxième joystick) donne les valeurs :

Bits	Valeurs	Effets
0	1	haut
1	2	bas
2	4	gauche
3	8	droite
4	16	feu
5	32	feu

Les valeurs s'ajoutent en cas de conditions combinées.

4b. – Ce connecteur permet de diriger le son vers votre chaîne stéréo. Le canal gauche reçoit les voix 1 et 3, le canal droite reçoit les voix 2 et 3.

5. – Le connecteur (34 broches) imprimante permet de relier n'importe quelle imprimante au standard Centronics. Attention, le connecteur de l'Amstrad n'est pas lui-même au standard Centronics. Il faudra vous procurer le câble adéquat.

Aussi, toute imprimante au standard Centronics est utilisable, mais seule une imprimante Amstrad vous donnera les caractères graphiques.

6. – Sur 664, ce connecteur (50 broches) est destiné à recevoir des extensions - notamment mémoire.

7. – Ce connecteur (34 broches) est, sur 664, destiné à recevoir une seconde unité de mini-disquettes. Sur 464, c'est sur le connecteur 6 que se relie l'unité de disquettes.

Nous arrêtons là cette description succincte.

Nous voici arrivés au terme de ce livre, au terme de notre découverte de l'Amstrad.

Il faut convenir que c'est là une merveilleuse machine, surtout en regard de son prix.

Ses possibilités sont immenses ; nous n'en avons découvert qu'une petite partie, tant en calcul, en gestion, en machine de loisirs ou d'éducation : notre programme de quiz géographique vous permet de réviser votre géographie en vous amusant... Les possibilités de traitement graphique sont spécialement utiles dans ce contexte, et spécialement perfectionnées sur l'Amstrad, ainsi que les possibilités musicales.

Enfin, nous avons esquissé comment l'Amstrad peut être étendu pour avoir encore plus de possibilités.

Nous laissons maintenant le champ libre à votre imagination pour en tirer le meilleur parti.

Annexe 1
FONCTIONS ET MOTS-CLÉS DU BASIC

Fonctions arithmétiques

L'application au 664 seule est signalée par un + ou la mention 664.

ABS	Valeur absolue de l'argument entre parenthèses.
ATN	Arc tangente - le résultat est en radians, compris entre −PI/2 et PI/2.
CINT	Conversion en entier.
COS	Cosinus - l'argument doit être en radians. *Exemple :* cos(x en degrés) = COS (PI*X/180).
CREAL	Conversion en réel.
DERR	Numéro d'erreur disque (664).
EOF	Indique si la fin du fichier indiqué est atteinte.
ERL	Numéro de ligne où s'est produite la dernière erreur.
ERR	Numéro de la dernière erreur.
EXP	Exponentielle e^x. l'argument doit être <=88, sinon il se produit un dépassement de capacité.
FIX	Partie entière. FIX(3.99) donne 3. FIX(−3.5) donne −3.

FRE — Quelle que soit la valeur de l'argument numérique, fournit le nombre d'octets restés libres en mémoire. Si l'argument est alphanumérique, déclenche une remise en ordre de l'espace des chaînes (garbage collection).

INKEY — Dit si la touche spécifiée est enfoncée.

INP — Lit un port d'entrée.

INT — Partie entière, ou plutôt plus grand entier inférieur ou égal à l'argument : INT(0.5) vaut 0; INT(5) vaut 5 ; INT(−0.5) vaut −1; INT(−3) vaut −3.

JOY — Lecture d'un manche à balai.

LOG — Logarithme naturel (népérien, ou en base e). Pour obtenir le logarithme de X en base Y, utiliser LOG(X)/LOG(Y).
Exemple : logarithme décimal de X = LOG(X)/LOG(10).

LOG10 — Logarithme décimal.

MAX — Maximum d'une série de nombres.

MIN — Minimum d'une série de nombres.

PEEK — Fournit le contenu (compris entre 0 et 255) de la case mémoire dont l'adresse est égale à l'argument (lequel doit être entier et compris entre 0 et 65535). Pour écrire en mémoire, voir POKE page 144.

POS — POS(#f) fournit la prochaine position d'impression libre sur la ligne dans la fenêtre f (position du curseur).

REMAIN — REMAIN(n) est le temps qui reste à courir dans le chrono n° n.

RND — Fournit un nombre pseudo-aléatoire compris entre 0 et 1. Voir l'explication plus détaillée au chapitre 5.

ROUND — Arrondit avec le nombre de décimales spécifiées.

SGN — Fonction "signe" : 1 si X > 0, −1 si X < 0 et 0 si X = 0.

SIN — Sinus - l'argument est supposé en radians.

SPC — Ne peut s'employer que dans une instruction PRINT. PRINT SPC(I) imprime I espaces. I doit être entier, compris entre 0 et 255.

SQ Indique l'état de la file d'attente sonore de la voix spécifiée.

SQR Racine carrée. L'argument doit être supérieur ou égal à 0.

TAB Ne peut s'employer que dans une instruction PRINT. PRINT TAB(I); fait aller à la position d'impression n° I (0 est la position la plus à gauche d'une ligne, 39 la plus à droite). I doit être compris entre 0 et 255. Si I>position où l'on est déjà, il n'y a pas d'action, c'est-à-dire que la prochaine impression se fera là où on était positionné.

TAN Tangente - l'argument est supposé en radians.

TEST Donne la couleur d'un point de l'écran graphique.

TESTR Donne la couleur d'un point de l'écran graphique spécifié en coordonnées relatives.

UNT Convertit une valeur 16 bits en entier en complément à 2, c'est-à-dire compris entre −32768 et +32767.

VPOS VPOS(#f) donne la position verticale du curseur dans la fenêtre f.

XPOS Position horizontale du curseur graphique.

YPOS Position verticale du curseur graphique.

Fonctions chaînes

ASC(X$) Code ASCII du caractère X$.

BIN$(X) Conversion en binaire.

CHR$(X) Caractère dont le code est X.

COPYCHR$+ Copie du caractère sous le curseur (664).

DEC$+ Conversion en décimal et formatage (664).

HEX$(X) Conversion en hexadécimal.

INKEY$ Caractère juste frappé au clavier.

INSTR(N,X$,Y$)	Position où, à partir de N, Y$ est présent dans X$.
LEFT$(X$,N)	N caractères les plus à gauche de X$.
LEN(X$)	Longueur de X$.
LOWER$(X$)	Met la chaîne argument en minuscules.
MID$(X$,K)	Caractères de X$ du K à la fin.
MID$(X$,K,N)	N caractères dans X$ à partir du K^{e}.
RIGHT$(X$,N)	N caractères les plus à droite de X$.
SPACE$(N)	Chaîne formée de N espaces.
STR$(X)	Conversion de X en chaîne de caractères.
STRING$(N,X$)	Chaîne formée de N fois le caractère X$.
STRING$(N,X)	Chaîne formée de N fois le caractère de code X.
UPPER$(X$)	Met la chaîne argument en majuscules.
VAL(X$)	Valeur représentée par la chaîne X$.

Mots-clés réservés du BASIC

ABS	AFTER	AND	ASC	ATN	AUTO
BIN$	BORDER	CALL	CAT	CHAIN	CHR$
CINT	CLEAR	CLG	CLOSEIN	CLOSEOUT	CLS
CONT	COPYCHR$$^{+}$$	COS	CREAL	CURSOR^{+}	DATA
DEC$$^{+}$$	DEF	DEFINT	DEFREAL	DEFSTR	DEG
DELETE	DERR^{+}	DI	DIM	DRAW	DRAWR
EDIT	EI	ELSE	END	ENT	ENV
EOF	ERASE	ERL	ERR	ERROR	EVERY
EXP	FILL^{+}	FIX	FN	FOR	FRAME^{+}
FRE	GOSUB	GOTO	GRAPHICS^{+}	HEX$	HIMEM
IF	INK	INKEY	INKEY$	INP	INPUT
INSTR	INT	JOY	KEY	LEFT$	LEN
LET	LINE	LIST	LOAD	LOCATE	LOG
LOG10	LOWER$	MASK^{+}	MAX	MEMORY	MERGE

MID$	MIN	MOD	MODE	MOVE	MOVER
NEXT	NEW	NOT	ON	ON BREAK	OPENIN
OPENOUT	OR	ORIGIN	OUT	PAPER	PEEK
PEN	PI	PLOT	PLOTR	POKE	POS
PRINT	RAD	RANDOMIZE	READ	RELEASE	REM
REMAIN	RENUM	RESTORE	RESUME	RETURN	RIGHT$
RND	ROUND	RUN	SAVE	SGN	SIN
SOUND	SPACE$	SPC	SPEED	SQ	SQR
STEP	STOP	STR$	STRING$	SWAP	SYMBOL
TAB	TAG	TAGOFF	TAN	TEST	TESTR
THEN	TIME	TO	TROFF	TRON	UNT
UPPER$	USING	VAL	VPOS	WAIT	WEND
WHILE	WIDTH	WINDOW	WRITE	XOR	XPOS
YPOS	ZONE				

MOTS-CLÉS DISQUES

La barre verticale est obtenue par SHIFT@

|A	|B	|CPM	|DIR	|DISC	|DISC.IN
|DISC.OUT	|DRIVE	|ERA	|REN	|TAPE	|TAPE.IN
|TAPE.OUT	|USER				

PSEUDO-VARIABLES

PI	Valeur de π
TIME	Horloge temps réel.

Annexe 2
RÉPERTOIRE DES INSTRUCTIONS ET DES OPÉRATEURS BASIC

Comme dans le Guide Michelin, nous attribuons des étoiles aux instructions et aux commandes : *** instruction fondamentale, ** instruction importante, * instruction intéressante. Pas d'* : instruction qu'il ne faut pas hésiter à utiliser si le besoin s'en fait sentir ; certaines de ces instructions ne sont pas détaillées dans ce livre.

D'autre part, P veut dire : plutôt instruction en mode programmé, P̸ veut dire : interdit en mode programmé, C : plutôt commande en mode direct, C̸ : interdit en mode direct, "rien" veut dire : tantôt l'un, tantôt l'autre.

Enfin, $^{+}$ signifie : cette instruction a été étendue sur 664/6128 et $^{++}$ signifie : instruction valable seulement sur 664/6128.

La définition est suivie d'exemples non commentés montrant les différentes formes que peut prendre l'instruction.

Catég.	Mot-clé	Définition **Exemples**	Page
	AFTER... ... GOSUB	Attend l'écoulement d'un délai **AFTER 250,1 GOSUB 1000**	
CP*	AUTO	Numérotation automatique **AUTO AUTO 100,20**	69
	BORDER	Définit la couleur du cadre **BORDER 5**	124
	CALL	Appel une routine en langage machine **CALL 0**	

Catég.	Mot-clé	Définition **Exemples**	Page
*	CAT	Donne le catalogue du disque ou de la casette	46
	CHAIN	Chaînage d'un programme : on charge le programme spécifié et on l'exécute **CHAIN"PROG"** **CHAIN"PROG.BAS",100**	
	CHAIN MERGE	Chaînage, mais avec fusion	
*	CLEAR	Vide les variables, abandonne les fichiers et met en mode radians **CLEAR**	33
	CLEAR INPUT	Écarte tous les caractères précédemment tapés au clavier	
*	CLG	Efface l'écran graphique et fixe la couleur du fond si elle est spécifiée **CLG CLG 4**	134
	CLOSEIN	Ferme un fichier ouvert en entrée sur disque ou cassette **CLOSEIN**	188
	CLOSEOUT	Ferme un fichier ouvert en sortie sur disque ou sur cassette **CLOSEOUT**	188
**	CLS	Vide l'écran (la fenêtre indiquée) **CLS**	33
CP*	CONT	Reprend l'exécution arrêtée par STOP ou ESC	65

Catég.	Mot-clé	Définition **Exemples**	Page
++	CURSOR	Rend le curseur visible (1) ou non (0) **CURSOR 1**	
P*	DATA	Définit une liste de constantes qui seront "lues" par une instruction READ **10 DATA ABC,DEF,5,3.25**	71
P*	DEF FN	Définition d'une fonction utilisateur **10 DEF FNF(X)=A*X+B**	129
P	DEFINT DEFREAL DEFSTR	Définit le type (entier, réel ou chaîne) d'un groupe de variables **DEFINT A-Z**	
	DEG	Passe en mode degrés pour les fonctions trigonométriques **DEG**	
CP	DELETE	Supprime un bloc d'instructions **DELETE 100-400**	70
	DI	Inhibe les interruptions	
P*	DIM	Dimensionnement d'un tableau (fixe les valeurs maximales des indices) **10 DIM A(15),B(3),C(2,5),D(7,6,2)** **DIM W(N)**	73
**	DRAW	Trace une ligne jusqu'au point indiqué de la couleur spécifiée **DRAW 100,200** **DRAW 100,200,5**	136

Catég.	Mot-clé	Définition **Exemples**	Page
	DRAWR	Trace une ligne jusqu'au point indiqué de la couleur spécifiée, mais en coordonnées relatives **DRAWR 100,200** **DRAWR 100,200,5**	136
*	EDIT	Édite la ligne indiquée **EDIT 100**	42
	EI	Rétablit les interruptions	
**	ELSE	Introduit l'instruction à effectuer lorsque la condition d'un IF n'est pas satisfaite **IF X<0 THEN A=B ELSE A=X**	52
P*	END	Termine un programme et ferme les fichiers	34
	ENT	Définit une modulation sonore (voir chapitre 8) **ENT 1,10,2,2**	153
	ENV	Définit une enveloppe sonore (voir chapitre 8) **ENV 1,10,2,2,10,0,2,20,−1,2**	151
	ERASE	Détruit les tableaux indiqués **ERASE A,B**	80
	ERROR	Produit fictivement une erreur **ERROR 5**	

Catég.	Mot-clé	Définition **Exemples**	Page
*	EVERY... ... GOSUB	Appelle un sous-programme périodiquement **EVERY 500,2 GOSUB 1000** Le 2e paramètre est le n° de chrono	101
*++	FILL	Remplit une aire fermée dans l'écran graphique avec l'encre spécifiée **FILL3**	136
**	FOR	Introduit une boucle : toutes les instructions comprises entre : FOR I=A TO B STEP C et le NEXT correspondant seront répétées pour toutes les valeurs de I allant de A à B, C par C **10 FOR I=1 TO 1000** **50 FOR I=0 TO 200 STEP 5** **60 FOR I=N TO 3*N+4 STEP 5** **70 FOR I=50 TO 1 STEP −1** **80 FOR X=1.5 TO 2*PI STEP.1**	57
++	FRAME	Synchronise l'action au balayage écran ce qui peut rendre l'affichage moins saccadé	
P**	GOSUB	Appel d'un sous-programme **10 GOSUB 1000**	118
P**	GOTO	Saute à une autre instruction **10 GOTO 50**	25
++	GRAPHICS PAPER	Établit la couleur de fond de l'écran graphique **GRAPHICS PAPER 3**	

Catég.	Mot-clé	Définition **Exemples**	Page
++	GRAPHICS PEN	Établit la couleur de tracé graphique et la transparence (2e paramètre = 1) **GRAPHICS PEN 1** **GRAPHICS PEN 1,1**	
P**	IF	Saut conditionnel, de la forme : IF condition THEN instruction Si la condition n'est pas satisfaite, on passe à la ligne suivante; si la condition est satisfaite, on effectue l'instruction qui suit THEN IF c THEN GOTO x s'élide en IF c THEN x ou IF c GOTO x **10 IF A>B THEN Y=K** **20 IF A=3 GOTO 1000** **30 IF A$<> ""THEN 50** **50 IF A<B THEN X=A ELSE X=B**	49
*	INK	Associe une couleur à un n° d'encre Si l'on spécifie deux couleurs, il y a clignotement **INK 1,0,13**	125
P***	INPUT	Acquisition de données au clavier **10 INPUT A 20 INPUT A,B,C$,D** **30 INPUT "Entrez un nombre";N** **50 INPUT #2,A**	24 27
P*	KEY	Définit une touche de fonction **KEY 128, "FOR I=1 TO N"**	40
P	KEY DEF	Change la signification d'une touche	
	LET	Instruction d'affectation. Ne pas employer ce mot, écrire A=...	

Catég.	Mot-clé	Définition **Exemples**	Page
	LINE INPUT	Accepte des caractères quelconques au clavier ou sur un fichier **LINE INPUT A$**	
C***	LIST	Liste du programme **LIST LIST 10– LIST –100** **LIST 10 LIST 10–100**	31
C***	LOAD	Chargement d'un programme sur cassette ou disquette **LOAD " " LOAD "PROG"**	46
P*	LOCATE	Positionne le curseur dans la fenêtre **LOCATE #5,20,10** **LOCATE 20,10**	99
++	MASK	Définit le pointillé utilisé pour les tracés **MASK &X00001111,0**	
	MEMORY	Met de la mémoire à l'abri de BASIC	
C*	MERGE	Fusionne un programme lu sur périphérique avec le programme présent en mémoire **MERGE "TOTO"**	70
**	MODE	Fixe le mode d'affichage et vide l'écran **MODE 2**	123

Catég.	Mot-clé	Définition **Exemples**	Page
*	MOVE	Déplace (sans tracé) le curseur graphique à l'emplacement indiqué **MOVE 100,200**	135
	MOVER	Comme MOVE, mais en coordonnées relatives **MOVER 10,10**	135
CP*	NEW	Vide la mémoire programme	33
**	NEXT	Fait passer à l'itération suivante dans un FOR **NEXT I NEXT J,I** **NEXT**	57
	ON GOTO GOSUB	**ON I GOTO 10,20,30** Si I vaut 1, on va en 10, s'il vaut 2, on va en 20, en 30 s'il vaut 3 **ON I GOSUB 1000,1500,2000,5000** Appelle le sous-programme en 1000, 1500, 2000 ou 5000 selon que I vaut 1, 2, 3 ou 4	
+	ON BREAK CONT GOSUB STOP	Dit ce que l'on doit faire si la touche ESC est enfoncée CONT : ne pas en tenir compte (664) GOSUB : aller au sous-programme indiqué STOP : arrêter comme d'habitude	
	ON ERROR GOTO	Va à l'instruction indiquée lorsqu'une erreur s'est produite **ON ERROR GOTO 1000**	

Catég.	Mot-clé	Définition **Exemples**	Page
	ON SQ(n) GOSUB	Appelle le sous-programme indiqué dès qu'il y a une place de libre dans la file d'attente du canal sonore spécifié **ON SQ(2) GOSUB 500**	
*	OPENIN	Ouvre un fichier en lecture **OPENIN "FICHIER"** Sur cassette, si le 1er caractère du nom est "!", les messages de démarrage du magnétophone sont supprimés	188
*	OPENOUT	Ouvre un fichier en écriture sur disque ou disquette **OPENOUT "FICHIER"**	188
*	ORIGIN	Fixe l'origine des coordonnées sur l'écran graphique et, éventuellement définit une fenêtre **ORIGIN 100,200** **ORIGIN X, Y, G, D, H, B**	134
	OUT	Envoie une donnée sur un port de sortie	
*	PAPER	Fixe la couleur de fond dans la fenêtre **PAPER 3 PAPER #5,3**	125
*+	PEN	Fixe la couleur d'écriture dans la fenêtre **PEN1 PEN #5,1 PEN #5,1,1** (664 : mode transparent)	125

Catég.	Mot-clé	Définition **Exemples**	Page
**+	PLOT	PLOT x,y,c,m trace le point (x,y) dans la couleur c et (664) avec le mode m m = 0 (normal), 1 (XOR), 2 (AND) ou 3 (OR) **PLOT 100,200 PLOT 100,300,2** (664) **PLOT 100,300,2,2**	135
**+	PLOTR	Comme PLOT, mais les coordonnées du point sont relatives **PLOTR 20,20**	135
*	POKE	POKE a,b écrit la donnée b à l'adresse a **POKE 36879,27** **POKE K+1,Z−4** Pour lire en mémoire, voir PEEK	144
***	PRINT	Imprime un résultat sur écran ou fichier **PRINT A** **10 PRINT A;B,J** **20 PRINT 2*A+3,B$** **30 PRINT "Le résultat est";B;** **PRINT #1,A** **PRINT USING "##.#";A**	25 28
	RAD	Passe en radians (c'est le mode normal) pour les fonctions trigonométriques	
	RANDOMIZE	Initialise le générateur de nombres aléatoires **RANDOMIZE TIME**	
P*	READ	"Lecture de données" dans une instruction DATA associée **10 READ A 20 READ A,B$,C**	71

Catég.	Mot-clé	Définition **Exemples**	Page
	RELEASE	Libère une voix sonore en attente **100 RELEASE 2**	150
P	REM	Introduit un commentaire	
P*	RENUM	Renumérote un programme **RENUM 500,100,10**	70
P	RESTORE	Revient au début des DATA **RESTORE RESTORE 1000**	72
	RESUME RESUME NEXT	Retour d'une routine de traitement d'erreur **RESUME 1000** **RESUME NEXT**	
P**	RETURN	Retour de sous-programme **100 RETURN**	118
C***	RUN	Déclenche l'exécution d'un programme **RUN RUN 500** **RUN "TOTO"**	33
C**	SAVE	Sauvegarde d'un programme sur cassette ou disquette **SAVE "TOTO"** **SAVE "ECRAN",B,C000,4000**	45
*	SOUND	Fait retentir un son (cf. chapitre 8) **SOUND 1,142,1000**	147
	SPEED INK	Règle la vitesse de clignotement des couleurs **SPEED INK 50,30**	

Catég.	Mot-clé	Définition **Exemples**	Page
	SPEED KEY	Règle le temps de latence et la vitesse par la répétition des touches	
	SPEED WRITE	Règle la vitesse d'écriture sur cassette *Conseil :* gardez SPEED WRITE 0	45
**	STEP	Introduit le pas d'incrémentation dans un FOR	59
£	STOP	Arrête le programme	65
	SYMBOL	Permet de redéfinir un dessin de caractère **SYMBOL 200,0,0,0,0,1,1,1,1**	112
	SYMBOL AFTER	Doit précéder une série de SYMBOL pour préciser à partir de quel code on fait des redéfinitions **SYMBOL AFTER 200**	112
	TAG	Envoie le texte à la position du curseur graphique	137
	TAGOFF	Annule l'effet de TAG	137
P**	THEN	Introduit l'instruction à effectuer quand un IF est satisfait	49
**	TO	Introduit la valeur limite dans un FOR	57
	TROFF	Arrête la trace	66

Catég.	Mot-clé	Définition **Exemples**	Page
*	TRON	Déclenche la trace (exécution avec affichage des n° de ligne où l'on passe)	66
	USING	Introduit un format dans un PRINT	187
	WAIT	Attend une certaine configuration sur port d'entrées-sorties	
**	WEND	Marque la fin d'une boucle introduite par WHILE	61
**	WHILE	Répète les instructions qui suivent jusqu'à WEND tant que la condition indiquée est vraie **WHILE NOT EOF** **100 WHILE N>0**	61
	WIDTH	Ajuste la largeur d'imprimante	188
	WINDOW	WINDOW #n,g,d,h,b définit la fenêtre n° n dans l'ordre gauche, droite, haut, bas **WINDOW #3,1,20,1,5**	143
	WINDOW SWAP	Échange le contenu de deux fenêtres **WINDOW SWAP 1,2**	144
	WRITE	Comme PRINT mais sépare les éléments par des virgules et met les chaînes de caractères entre guillemets **WRITE #9,A,B$**	

Catég.	Mot-clé	Définition **Exemples**	Page
	ZONE	Définit l'écart (normalement 13) entre les positions de tabulation (éléments séparés par "," dans PRINT **ZONE 10**	
***	=	Affectation d'une valeur à une variable **X=3*A+SIN(2*T) N=N+1**	18

Opérateurs

ARITHMÉTIQUES

+ addition de nombres ou concaténation de chaînes de caractères.
– Prendre l'opposé ou soustraction.
* Multiplication.
/ Division.
\ Division entière (11\4 donne 2).
MOD Modulo ou reste de division (11 MOD 4 donne 3).

DE RELATION

=	Égal.	<>	Différent.
>	Supérieur.	<=, =<	Inférieur ou égal.
<	Inférieur.	>=, =>	Supérieur ou égal.

LOGIQUES

(effectués bit à bit)

NOT Non logique, agit sur un seul opérande.
AND Et logique.
OR Ou logique. } 2 opérandes
XOR Ou exclusif.

bit	NOT	bit 1	bit 2	AND	OR	XOR
0	1	0	0	0	0	0
1	0	0	1	0	1	1
		1	0	0	1	1
		1	1	1	1	0

Annexe 3
MESSAGES D'ERREUR

Le fait de disposer d'un interpréteur qui prend les instructions BASIC une par une pour les exécuter permet, lorsque se produit une erreur, d'identifier avec précision l'instruction où s'est produit l'incident.

Par suite, les messages seront de la forme :

message in numéro

précisant le numéro de la ligne où l'erreur a été rencontrée. Le seul cas où le numéro n'est pas précisé est celui d'une commande directe.

Les problèmes les plus courants proviennent d'un mot-clé mal orthographié, d'une virgule manquante ou d'une variable d'un certain type employée à tort, etc. Il faut noter qu'une erreur signalée à un numéro de ligne peut n'être que la conséquence d'une autre erreur : par exemple, une erreur Division by zéro vient plutôt des instructions précédentes où le diviseur est calculé.

Nous donnons ci-dessous la traduction et, pour les plus importants, des explications des différents messages d'erreur qui peuvent être délivrés par les Amstrad lors de l'exécution d'un programme BASIC. On indique aussi le numéro de l'erreur (donné par la fonction ERR).

Rappelons que pour les erreurs auxquelles cela convient, le message d'erreur est suivi de l'impression de l'instruction concernée prête à être éditée.

Array already dimensioned - Tableau déjà dimensionné (10)

On est repassé une deuxième fois sur l'instruction DIM portant sur un tableau, ou il y a une deuxième instruction DIM pour un tableau.

Broken in - Appui sur ESC pendant une opération disque (32)

Cannot continue - On ne peut pas continuer (17)

Impossibilité de reprendre l'exécution d'un programme par CONT. C'est le cas s'il y a eu une erreur, ou si, pendant l'interruption, on a modifié ou ajouté une instruction. On peut reprendre par GOTO numéro dans certains cas.

DATA exhausted - DATA épuisées (4)

On essaie de faire un READ alors qu'on a déjà "lu" toutes les données de toutes les instructions DATA. Il faut, soit comptabiliser les données avec soin, soit utiliser RESTORE.

Direct command found - Commande directe mal placée (21)

On rencontre une ligne sans numéro lors de la lecture d'un programme sur cassette ou disquette.

Division by zero - Division par zéro (11)

Vient le plus souvent d'une variable non initialisée.

EOF met - Fin de fichier (24)

On rencontre une fin de fichier sur cassette ou disquette et l'on recherche à lire plus loin.

File already open - Fichier déjà ouvert (27)

On exécute une instruction OPENIN ou OPENOUT avant que le fichier précédent n'ait été fermé.

File not open - Fichier non ouvert (31)

Essai d'accéder à un fichier avant d'exécuter un OPENIN ou OPENOUT.

File type error - Erreur dans le type d'un fichier (25)

Le fichier n'est pas du type voulu. OPENIN ne peut s'adresser qu'à un fichier texte ; LOAD ne peut charger qu'un fichier créé par SAVE.

Improper argument - Valeur erronée de l'argument (5)

Emploi d'une fonction avec un argument hors de l'intervalle permis.

Exemples :

- indice négatif ou > 32767 ;
- argument de LOG négatif ou nul ;
- argument de SQR négatif ;
- 0 puissance nombre négatif ;
- longueur spécifiée dans MID$, RIGHT$ ou LEFT$ non comprise entre 0 et 255 ;

- index de ON GOTO hors des limites ;
- adresse dans PEEK, POKE, non comprise entre 0 et 65535 ;
- octet spécifié dans POKE, TAB, SPC, etc., non compris entre 0 et 255.

Invalid direct command - Illégal en mode direct (12)

INPUT et DEF FN sont interdites en mode direct.

Lines does not exist - Ligne non définie (8)

DELETE, GOTO, GOSUB ou THEN renvoyant à un numéro de ligne inexistant. C'est le plus souvent dû à l'oubli de correction d'un GOTO... quand la ligne cible a été supprimée ou changée de place.

Line too long - Ligne trop longue (23)

Une ligne BASIC ne doit pas dépasser 255 caractères.

Memory full - Mémoire épuisée (7)

Programme trop long, ou trop de variables, ou trop de boucles et GOSUB imbriqués.

NEXT missing - Il manque le NEXT (26)

On ne rencontre pas le NEXT qui devrait correspondre au dernier FOR rencontré.

Operand missing - Manque d'un opérande (24)

Opérateur non suivi d'opérande dans une expression arithmétique ou absence d'un paramètre obligatoire dans une instruction.

Overflow - Dépassement de capacité (6)

Résultat d'un calcul supérieur à 1.7 E38. Dans l'autre sens, un résultat inférieur à 1. E−38 est indiscernable de 0, mais il n'y a pas de message d'erreur.

Redo from start - Recommencez depuis le début

Lors d'une instruction INPUT, on a fourni une quantité alphanumérique alors qu'on s'attendait à du numérique. Il faut reprendre en redonnant toutes les valeurs attendues par l'instruction INPUT considérée.

RESUME missing - Pas de RESUME (21)

Pas de RESUME alors qu'on est dans une routine de traitement d'erreur.

String expression too complex - Expression chaîne de caractères trop complexe (16)

String space full - Débordement de l'espace des chaînes (14)

On dépasse la quantité de mémoire permise pour les chaînes de caractères même après "garbage collection"

String too long - Chaîne de caractères trop longue (15)

Tentative de créer (par concaténation) une chaîne de caractères de plus de 255 caractères.

Subscript out of range - Indice hors des limites (9)

Tentative d'appeler un élément de tableau d'indice supérieur à la limite fournie dans le DIM, ou encore avec un nombre d'indices différent de celui spécifié dans le DIM. *Exemple :* DIM A(15) avec A(20) ou A(2,2).

Syntax error - Erreur de syntaxe (2)

Instruction incompréhensible pour BASIC. Dû notamment à des parenthèses non appariées, caractères illégaux, mauvaise ponctuation, faute d'orthographe dans un mot-clé.

Type mismatch - Désaccord entre numérique et alpha (13)

Valeur numérique affectée à une variable chaîne, ou vice versa, ou bien argument numérique fourni à une fonction qui demande un argument alphanumérique ou vice versa.

Unexpected NEXT - NEXT sans FOR correspondant (1)

Vient le plus souvent de boucles mal imbriquées, d'une confusion sur la variable marquée dans le NEXT ou encore si l'on a supprimé un FOR lors d'une correction, en oubliant de supprimer le NEXT correspondant.

Unexpected RESUME - RESUME sans erreur (22)

On a rencontré une instruction RESUME alors qu'on n'a pas appelé de routine de traitement d'erreur.

Unexpected RETURN - RETURN sans GOSUB (3)

On est tombé sur un RETURN alors que l'on ne venait pas d'exécuter un GOSUB. Dû le plus souvent à l'oubli de END en fin de programme principal qui précède un sous-programme.

Unexpected WEND - WEND sans WHILE (30)

On tombe sur un WEND sans avoir trouvé le WHILE correspondant il peut avoir été supprimé lors d'une modification de programme alors qu'on a oublié de supprimer le WEND).

Unknown command - Commande externe inconnue (28)

Unknown user function -Fonction non définie (18)

Référence à une fonction pour laquelle on a oublié le DEF FN.

WEND missing - Il manque un WEND (29)

On ne trouve pas le WEND correspondant à un WHILE ouvert.

Les autres erreurs concernent les disquettes.

Annexe 4
QUESTIONS ET RÉPONSES

Lorsque je veux relancer un programme après interruption (en 100), je peux utiliser CONT, RUN 100 ou GOTO 100 (en mode direct). Quelle est la différence?

CONT ne peut pas être utilisé si, au cours de l'interruption, vous avez apporté une correction au programme. Si vous utilisez RUN 100, toutes vos variables seront remises à zéro. Donc, si lors de l'interruption, vous avez corrigé une variable, il faut utiliser GOTO 100.

Lorsque la variable à lire par INPUT est une chaîne de caractères, doit-on mettre la valeur entre guillemets?

En principe non. On la met entre guillemets si la chaîne contient des virgules, deux-points, ou des espaces au début ou à la fin.

Que se passe-t-il lorsqu'en INPUT d'un nombre, l'on fournit une chaîne de caractères, ou vice versa?

Si à INPUT A$ vous répondez 123, l'Amstrad prendra la chaîne de caractères chiffre 1, chiffre 2, chiffre 3; pas de problème.

Si à INPUT A vous répondez ABC, il y aura une erreur avec le message Redo from start et l'instruction INPUT sera réexécutée; donc, pour un INPUT à plusieurs variables, vous devrez refournir toutes les données.

Y a-t-il une limite à la longueur des chaînes de caractères?

Oui : 255 caractères, ce qui est déjà beaucoup. Notez qu'une telle chaîne ne peut être donnée d'un seul coup, puisqu'une instruction ne peut dépasser 255 caractères; elle doit être construite par concaténation.

Comment prévoir la taille d'un programme?

En gros, un programme occupe autant d'octets qu'il renferme de caractères, puisqu'il est stocké tel quel, comme chaîne de caractères, sauf les mots-clés

qui sont remplacés par un code en 1 octet. Il y a aussi la place des numéros de ligne.

En ce qui concerne les variables, chaque variable numérique occupe 9 octets, chaque chaîne occupe (9+longueur) octets. Un tableau occupe x(n+1)+2d octets, où d est le nombre de dimensions, n est la taille du tableau (y compris l'élément nx 0) et x=7 (nombres) ou 3 (chaînes de caractères).

On gagne de la place mémoire en supprimant tous les blancs inutiles, en mettant plusieurs instructions par ligne, en évitant les REM, en utilisant des variables plutôt que des constantes. Utilisez des GOSUB dès qu'il faut faire appel plusieurs fois à une séquence d'instructions identiques.

J'ai des problèmes lorsque j'ai plusieurs instructions sur la même ligne, avec un IF parmi elles.

La forme

```
10 inst.1 : inst.2 : IF c THEN inst.3
20 inst.4
```

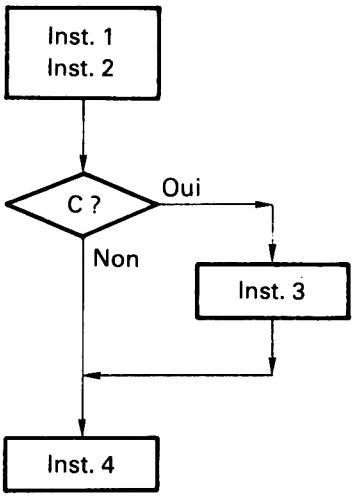
ne doit vous causer aucun problème : elle obéit à l'ordinogramme *a*) ci-dessous.

La forme

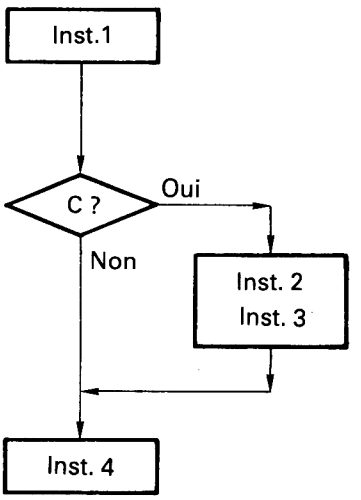
```
5 inst.1
10 IF c THEN inst.2 : inst.3
20 inst.4
```

obéit, *sur Amstrad,* à l'ordinogramme *b*) ci-dessous. C'est-à-dire que lorsque la condition n'est pas satisfaite, on passe à la ligne suivante et non à l'instruction qui suit l'instruction introduite par THEN.

a)



b)



Que se passe-t-il si je mets une instruction FOR sans NEXT?

Vous aurez le diagnostic : NEXT missing in...

Comment imprimer des nombres alignés sur leur droite?

Il faut employer les fonctions chaînes de caractères. Par exemple, on dispose d'une zone de 10 caractères dans laquelle on veut imprimer un nombre N. Quel que soit le nombre de chiffres, on veut que le nombre soit imprimé à droite de la zone. On emploie la séquence suivante :

```
100 N$=STR$(N) :L=LEN(N$)
110 IF L=10 GOTO 130
120 FOR I=1 TO 10-L :N$=" "+N$ :NEXT
130 PRINT N$
```

On peut remplacer 120 par 120 PRINT SPC(10-L);

On peut aussi procéder par PRINT USING qui formate les données; ici,il suffirait de :

```
100 PRINT USING "##########";N
```

Enfin sur un 664, on peut utiliser la fonction DEC$.

Je viens de faire un programme que je voudrais sauvegarder sur cassette, mais j'ai oublié de positionner la cassette.

Il ne faudrait surtout pas faire le SAVE avec la cassette rembobinée : vous effaceriez une partie des programmes déjà enregistrés. Utilisez CAT pour faire dévider la cassette jusqu'à la zone vierge.

Comment savoir ce qu'il y a sur une cassette?

Il faut tenir le répertoire des cassettes à mesure qu'on les remplit. Sinon, utilisez la commande CAT sur le moment.

J'ai vu, sur certains listings, des noms de variables se terminant par %. Qu'est-ce que c'est?

Il s'agit d'un type de variable que nous n'avons pas étudié : les variables entières. Leurs noms sont de la forme A%, A1%, ALBERT%. On peut, dans un programme, avoir les trois variables distinctes A%, A et A$. Les variables % peuvent être dimensionnées. Ces variables ne peuvent prendre que des valeurs entières comprises entre −32767 et +32767. Elles sont intéressantes pour économiser de la mémoire, étant donné qu'elles n'occupent que deux octets par élément, dans un tableau.

Est-ce que, à part des programmes, l'on peut écrire des données sur cassette?

Oui. Bien que cette question fasse partie de l'ensemble des traitements de fichiers et dépasse le sujet de ce livre, nous vous donnons quelques éléments.

Pour écrire des données :

– on doit tout d'abord ouvrir un fichier

OPENOUT "NOM"

↑
nom de fichier géré comme un nom de programme

– ensuite, donner les ordres d'écritures successives des données, par une instruction très semblable à PRINT dans une fenêtre mais avec le n° 9 :

PRINT #9,A

↑
variable à écrire

– enfin, fermer le fichier :

CLOSEOUT

Pour lire :

– ouverture

OPENIN "NOM"

– ordres de lectures successives INPUT #9,A (analogue à l'INPUT clavier, mais comme si le n° de la fenêtre était 9).

– fermeture

CLOSEIN

J'ai une imprimante. Comment l'utiliser?

C'est comme si le n° de la fenêtre était 8.

1° Pour obtenir un listing sur imprimante, faire :

LIST #8 ou LIST 100 – 200,#8

2° Pour écrire sur imprimante, faire PRINT #8 en lieu et place de PRINT.

On fixe la largeur de ligne par WIDTH largeur.

J'ai une unité de disquettes. Comment l'utiliser ?

Que ce soit l'unité incorporée au 664 ou une unité ajouté à un 464, vous l'utilisez exactement comme la cassette, avec le "n° de fenêtre 9".

Les instructions CAT, OPENIN, OPENOUT, SAVE, LOAD, MERGE, PRINT #9, INPUT #9, etc., iront sur la cassette si vous avez fait :

|TAPE, |TAPE.IN (entrée seulement) ou |TAPE.OUT (sortie)

Elles iront sur disquette si vous avez fait :

|DISC, |DISC.IN ou |DISC.OUT

C'est le cas par défaut si l'unité de disquette est présente.

Annexe 5
SOLUTION DES EXERCICES

Les exercices non rappelés ici ont, en principe, leur solution dans le texte.

Exercice 2.1.

```
5 PI=3.1415926535
10 INPUT "RAYON,HAUTEUR";R,H
20 V=PI*R ↑ 2*H
30 PRINT "VOLUME =" ;V
40 GOTO 10
```

20 et 30 pourraient être remplacés par :

```
25 PRINT "VOLUME =" ;PI*R ↑ 2*H
```

Exercice 2.2.

```
1 REM  Ex 2-2
2 REM
10 INPUT "CAPITAL, TAUX EN %, NB. D'ANNEES ";C,T
,N
20 I=C*((1+T/100)^N-1)
30 PRINT "INTERET = ";I
40 GOTO 10
```

Exercice 3.2.

Appuyez sur DEL deux fois, puis JOUR.

Exercice 3.4.

Pour que l'instruction 30 passe de

```
30 PRINT "SURFACE =" ;S
```

à

```
30 PRINT "VOLUME =" ;V
```

amener le curseur de copie sur le S de SURFACE et taper

VOLUME = ";V ENTER.

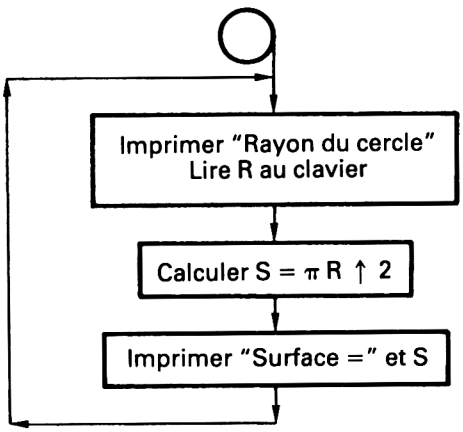
Exercice 4.1.

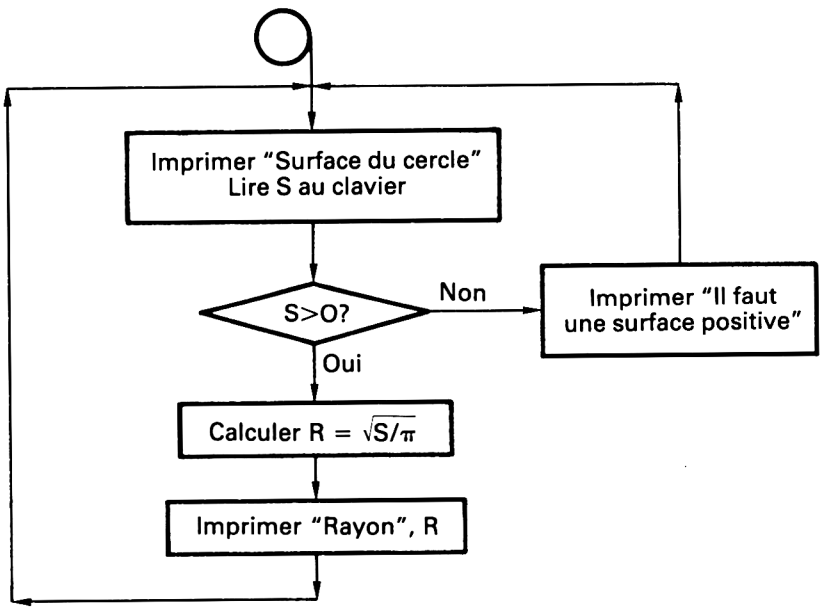
```
15 IF S<=0 GOTO 10
```

variante :

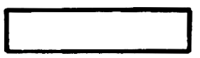
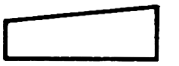
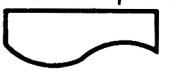
```
15 IF S<=0 GOTO 50
50 PRINT "Il faut une surface positive"
60 GOTO 10
```

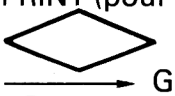
Exercice 4.2.





Exercice 4.3.

 calcul, INPUT (pour lequel on emploie parfois ) ou PRINT (pour lequel on emploie parfois  ou ).

 IF.

 GOTO ou IF.

 END ou dernière instruction.

 première instruction ou numéro cité dans la commande RUN ou GOTO en mode direct.

Exercice 4.4.

```
PRINT INT(X*10 ↑ D)/10 ↑ D
```

Exercice 4.5.

Deux solutions :

1. faire l'impression en 65 et mettre 60 N=N+1 ;

2. initialiser N à 1, ce qui doit se faire explicitement en ajoutant 10 N=1.

Exercice 4.7.

Il manque l'impression d'une ligne de titre!

```
5 PRINT"NB" ;"CARRE" ;"RACINE"
6 PRINT
```

(le 6 fait sauter une ligne)

Exercice 4.8.

Seule instruction changée :

```
10 FOR N=2 TO 10 STEP 2
```

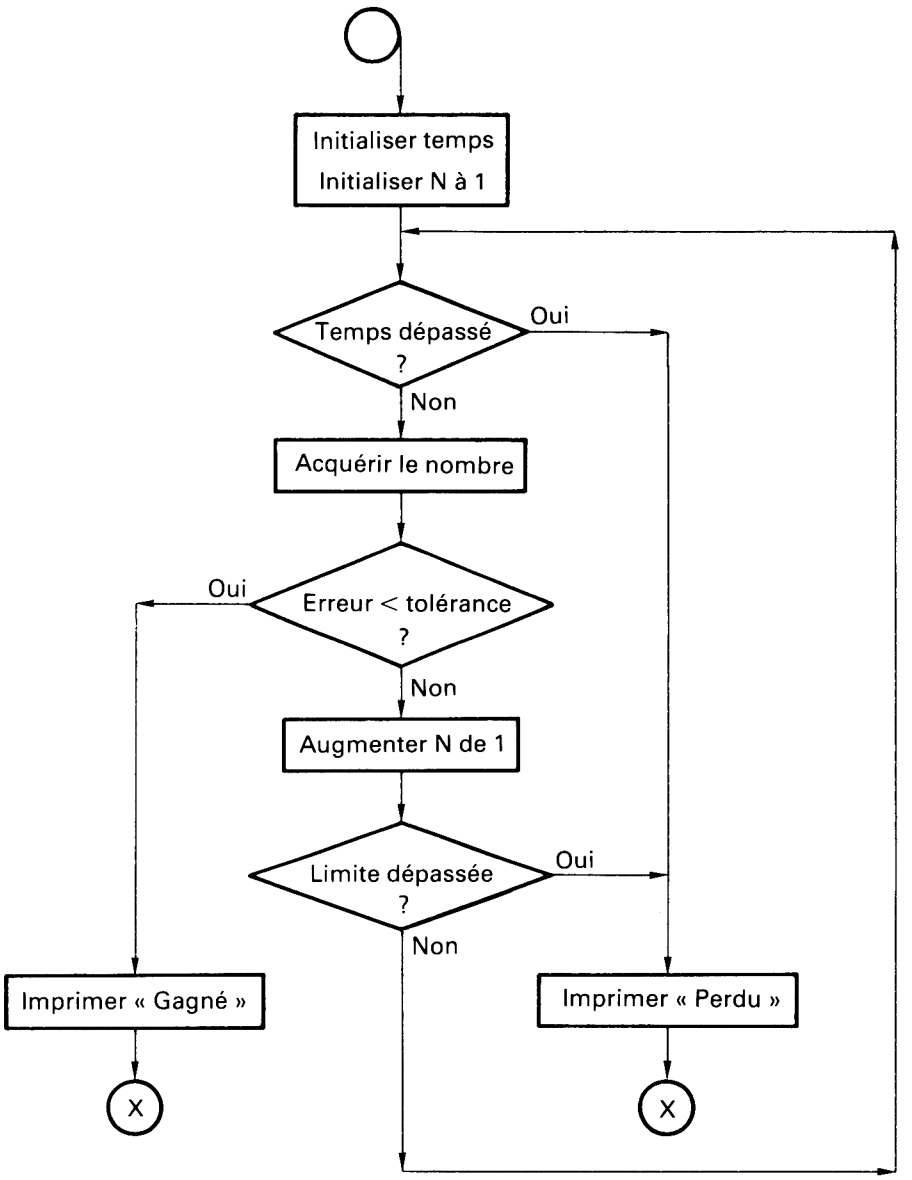
Exercice 4.9.

```
10 : 1 3 5 7          valeur finale 9
50 : 10 9 8 7 6 5 4   valeur finale 3
```

Exercice 4.11.

2 minutes.

Exercice 4.12.



Exercice 4.13.

Problème déjà résolu dans l'instruction 40 ; 60 devient :

60 PRINT "Gagné en" ;N+1 ; "coups et" ; INT((TIME−T)/3))/100 ; "secondes"

Exercice 4.14.

```
1 REM  Ex 4-14
2 REM
15 IF (TIME-T)/300>120 GOTO 55
50 PRINT "Nombre d'essais permis depasse"
53 END
55 PRINT "Temps depasse"
57 END
```

Exercice 5.1.

```
1 REM  Ex 5-1
2 REM
25 IF C<>99999 GOTO 30
27 RESTORE :GOTO 20
100 GOTO 10
210 DATA 210,78.31,901.5,31,4.18,2.7,99999
```

Exercice 5.2.

A la suite du calcul de la moyenne, on aurait :

```
1 REM  Ex 5-2A
2 REM
90 V=0
100 FOR I=1 TO N
110 V=V+(A(I)-M)^2
120 NEXT I
130 V=V/(N-1)
```

On peut proposer une solution plus élégante qui calcule moyenne et variance dans la même boucle et qui repose sur la remarque mathématique suivante :

$$\begin{aligned}\Sigma_i (A_i - M)^2 &= \Sigma_i (A_i^2 - 2\,MA_i + M^2) \\ &= \Sigma_i A_i^2 - 2\,M\Sigma_i A_i + NM^2 \\ &= \Sigma_i A_i^2 - NM^2\end{aligned}$$

D'où le programme :

```
1 REM  Ex 5-2B
2 REM
10 DIM A(N)
20 REM Lecture des A
30 S=0:S2=0
40 FOR I=1 TO N
50 S=S+A(I):S2=S2+A(I)^2
60 NEXT I
70 M=S/N:V=(S2-N*M^2)/(N-1)
80 PRINT "Moyenne";M;"  Variance";V
```

Exercice 5.3.

```
1 REM  Ex 5-3
2 REM
10 DIM U(N),V(N)
20 REM Lecture
30 UV=0
40 FOR I=1 TO N
50 UV=UV+U(I)*V(I)
60 NEXT I
70 PRINT UV
```

Exercice 5.5.

```
1 REM  Ex 5-5
2 REM
10 REM On ne s'occupe pas du tout
15 REM des Entrees/Sorties
20 DIM A(N,N),B(N,N),C(N,N)
30 FOR I=1 TO N
40 FOR J=1 TO N
50 C(I,J)=0
60 FOR K=1 TO N
70 C(I,J)=C(I,J)+A(I,K)*B(K,J)
80 NEXT K:NEXT J:NEXT I
```

Exercice 5.6.

```
10 X$=LEFT$(X$,4)+"A"+MID$(X$,6)
```

Exercice 5.7.

```
1 REM  Ex 5-7
2 REM
100 N=LEN(A$):P=LEN(B$)
110 FOR K=1 TO N-P+1
120 IF MID$(A$,K,P)=B$ GOTO 150
130 NEXT K
140 K=0
150 PRINT K
160 END
```

Lorsque vous aurez vu les sous-programmes, vous comprendrez qu'il est très judicieux de remplacer 160 par :

```
160 RETURN
```

Exercice 5.8.

NC=LEN(STR$(A))−1

dans les deux cas.

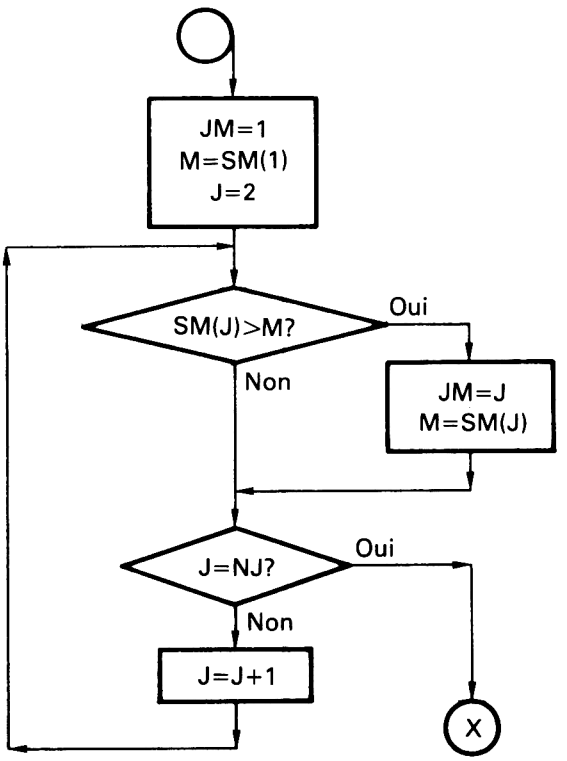
Exercice 5.9.

```
100 B$=STR$(B) :N=LEN(B$)
110 PRINT LEFT$(B$,N−3)
```

Exercice 5.10.

On suppose au départ que le maximum est au rang 1.

Ensuite, on parcourt le tableau SM à partir du rang 2. Si l'on trouve un élément supérieur au maximum supposé, c'est cet élément que l'on prend comme maximum supposé.



On peut alors ajouter au programme B-10 la séquence suivante :

```
208 REM  Ex 5-10
209 REM
210 PRINT:PRINT:DIM SM(NJ)
220 FOR J=1 TO NJ:SM(J)=0
230 FOR P=1 TO NP:SM(J)=SM(J)+SC(J,P)
240 NEXT P:SM(J)=SM(J)/NP:NEXT J
250 JM=1:M=SM(1)
260 FOR J=2 TO NJ
270 IF SM(J)<=M THEN 290
280 JM=J:M=SM(J)
290 NEXT J
300 PRINT"Le gagnant est ";NOM$(JM)
310 PRINT"avec un score moyen de";M
```

Exercice 5.11.

Les problèmes de classement ou de tri sont parmi les plus souvent rencontrés. On distingue les problèmes de tri simple où, partant d'un tableau d'éléments, on cherche à obtenir, à la fin du programme, le même tableau, mais ordonné, les éléments étant maintenant croissants ou décroissants; et les problèmes de classement où on laisse en place les éléments du tableau de départ, mais on forme un tableau auxiliaire, dit tableau de pointeurs PO(I), tel que PO(I) soit l'indice dans le tableau de départ de l'élément qui mérite d'être classé I^{e}. Le nom du joueur classé I^{e} est ainsi NOM$(PO(I)).

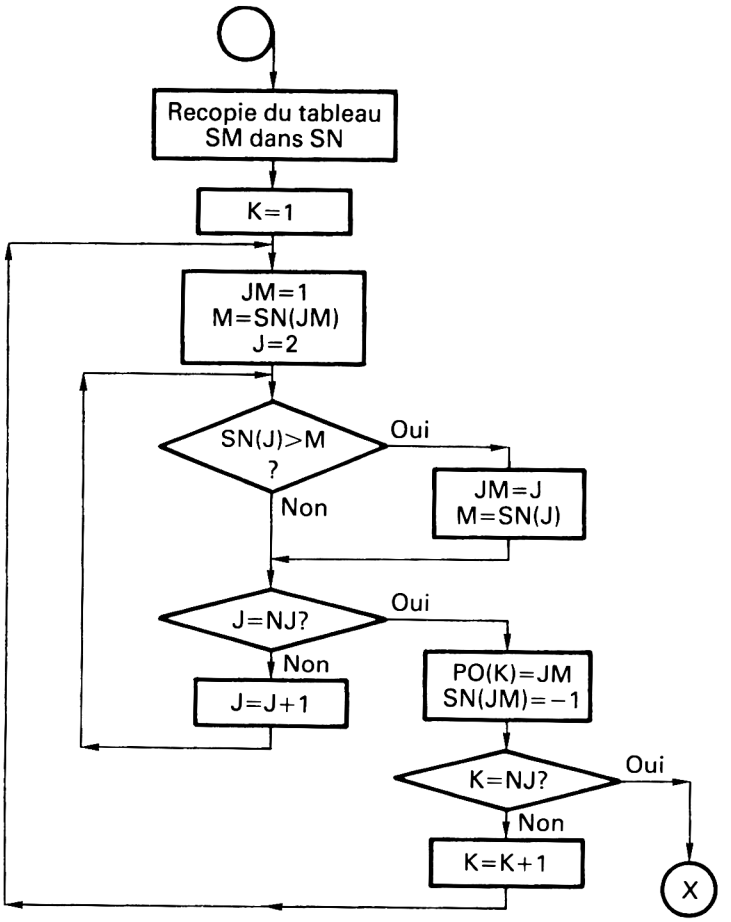
Dès que les éléments sont encombrants, il est plus intéressant d'effectuer un classement, plutôt qu'un tri simple : les termes à déplacer sont plus petits.

Une méthode de classement simple découle de l'exercice précédent. Supposons qu'on veuille un classement par ordre de score moyen décroissant. PO(1) n'est rien d'autre que le JM obtenu précédemment. PO(2) n'est autre que le rang du maximum suivant et ainsi de suite. Mais il y a un problème.

Lorsqu'on cherche le maximum d'un certain rang, il ne faut pas reprendre un maximum obtenu précédemment. Donc, lorsqu'on trouve un maximum, il faut remplacer l'élément par une valeur inférieure à toute valeur possible (ici −1) pour que l'élément ne soit plus repris. Dans ce cas, le tableau SM sera détruit par le classement. Pour l'éviter, on recopie d'abord le tableau dans un autre tableau, SN, sur lequel on travaillera.

D'où l'ordinogramme ci-après qui n'est que la répétition de celui de l'exercice 5.10. pour chaque valeur de K.

K représente le rang, à un instant donné, dans le classement.




```
248 REM  Ex 5-11
249 REM
250 DIM SN(NJ),PO(NJ)
260 FOR J=1 TO NJ: SN(J)=SM(J): NEXT
270 FOR K=1 TO NJ
280 JM=1:M=SN(JM)
290 FOR J=2 TO NJ
300 IF SN(J)<=M THEN 320
310 JM=J: M=SN(J)
320 NEXT J
330 PO(K)=JM: SN(JM)=-1
340 NEXT K
350 PRINT "Classement":PRINT
360 PRINT"Rang  Joueur  Score moyen":PRINT
370 FOR I=1 TO NJ
380 PRINT I;"    ";NOM$(PO(I));"     ";SM(PO(I)):N
EXT
```

Il existe d'autres méthodes de tri. Une des plus connues s'appelle méthode du "tri à bulles". On parcourt le tableau à classer en comparant à chaque fois deux éléments consécutifs. S'ils sont dans le bon sens, on les laisse; s'ils sont dans le mauvais ordre, on les échange.

Si, lors d'un parcours, il y a eu au moins un échange, on fait un nouveau parcours. S'il n'y a eu aucun échange, c'est que, maintenant, le tableau est ordonné.

Les méthodes de classement s'appliquent au classement alphabétique des chaînes de caractères, puisque IF A$<B$ est vrai si A$ précède B$. La séquence suivante classe par ordre alphabétique le tableau des noms NOM$:

```
390 REM  Ex 5-11   2e partie
391 REM
400 ECH=0: REM Indicateur d'echange
410 FOR I=1 TO NJ-1
420 IF NOM$(I)<=NOM$(I+1) THEN 450
430 T$=NOM$(I+1):X=SM(I+1): REM Sauvegarde pour
echange
440 NOM$(I+1)=NOM$(I):NOM$(I)=T$:SM(I+1)=SM(I):S
M(I)=X:ECH=1
450 NEXT I
460 IF ECH=1 GOTO 400
470 PRINT:PRINT "Classement alphabetique"
480 FOR I=1 TO NJ:PRINT NOM$(I);"    ";SM(I):NEXT
 I
```

Exercice 6.1.

On utilise des caractères de code 150 et 153.

Exercice 6.3.

```
200 LOCATE 17,13,0 : PRINT H,M,S : GOTO 200
```

Exercice 6.4.

```
88 REM  Ex 6-4
89 REM
90 PRINT "On recommence ?"
91 A$=INKEY$: IF A$="" GOTO 91
92 IF A$="o" GOTO 10
93 IF A$="n" THEN END
94 GOTO 91
```

Exercice 6.5.

```
100 A$=INKEY$ : IF A$= ""GOTO 100
... suite...
```

Exercice 6.6.

Les principales étapes de la solution sont :

1. Décider des caractères à employer pour représenter le coucou. Nous proposons le cœur - code 228 - et, pour les pattes et le support, la fusée de code 239.

2. Trouver les adresses écran pour mettre le coucou dans l'encadrement de la porte. On voit, d'après les ordres d'impression, que le cœur sera en 16e ligne, 8e colonne, d'où les LOCATE en 523 et 525. Le mieux est de les déterminer à tâtons.

D'où le programme C-4 :

```
1 REM  Programme C-4
2 REM
10 INPUT "HEURE (h,m,s)";H,M,S:CLS
20 LOCATE 1,7:PRINT
30 READ A: IF A=9999 GOTO 80
40 IF A>0 THEN PRINT CHR$(A);: GOTO 30
50 B=-A: READ A
60 FOR I=1 TO B: PRINT CHR$(A);
70 NEXT : GOTO 30
80 EVERY 50 GOSUB 500
90 LOCATE 9,21:PRINT H;M;S;"     ":GOTO 90
100 DATA -14,32,140,13,10,-10,32,204,-3,208,131,
208,205,13,10
110 DATA -9,32,204,-7,32,205,13,10
120 DATA -8,32,204,-9,210,205,13,10
130 DATA -9,32,211,-7,32,209,13,10
140 DATA -9,32,211,204,208,205,32,150,158,156,20
9,13,10
150 DATA -9,32,211,211,32,209,32,151,159,157,209
,13,10
160 DATA -9,32,211,211,32,209,32,147,155,153,209
,13,10
170 DATA -9,32,211,211,32,209,-4,32,209,13,10
180 DATA -9,32,211,211,32,209,-4,32,209,13,10
190 DATA -8,32,-11,208,13,10
200 DATA 9999
500 S=S+1:IF S<60 THEN 530
510 S=0:M=M+1:IF M<60 THEN 530
520 M=0:H=H+1
522 FOR K=1 TO 6
523 LOCATE 12,15:PRINT CHR$(228):LOCATE 12,16:PR
INT CHR$(239)
524 FOR T=1 TO 200:NEXT
525 LOCATE 12,15:PRINT" ":LOCATE 12,16:PRINT" "
526 FOR T=1 TO 100:NEXT
527 NEXT
528 LOCATE 9,21
530 RETURN
```

– Le test des heures rondes est évident : en 520, on est à une heure ronde par définition.

– En 523, on amène le coucou.

– En 524, on maintient le coucou pendant un certain délai (environ un dixième de seconde).

– En 525, on éteint le coucou (en inscrivant des espaces).

– En 526, on maintient le coucou éteint pendant un délai moitié du délai précédent.

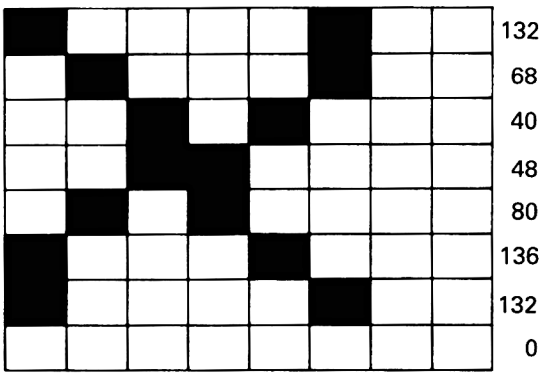
On fait une suite de 6 apparitions-extinctions. On peut en faire plus et on peut jouer sur les délais.

On peut aussi faire apparaître le coucou tous les quarts d'heure. Il ne manque plus qu'une commande sonore!

Exercice 6.7.

```
1 REM  Ex 6-7
2 REM
10 SYMBOL AFTER 65
20 SYMBOL 65,132,68,40,48,80,136,132,0
30 PRINT:PRINT "A A A A"
```

On utilise la grille :



Nous avons défini les valeurs des lignes en décimal, mais il serait certainement plus facile de les définir d'abord en binaire. Pour la première ligne, on aurait 100001000 = 132.

Exercice 6.8.

Cette fois, on remplace les caractères A à J.

```
1 REM  Ex 6-8  Corse
2 REM
10 SYMBOL AFTER 65
20 SYMBOL 65,0,0,0,0,0,0,0,1
25 SYMBOL 66,2,3,3,7,6,38,126,255
30 SYMBOL 67,11,15,15,31,63,15,7,63
35 SYMBOL 68,255,255,255,255,255,255,255,255
40 SYMBOL 69,128,192,192,192,192,192,192,128
45 SYMBOL 70,31,31,7,15,31,31,3,7
50 SYMBOL 71,255,255,255,254,252,252,252,252
55 SYMBOL 72,128,128,128,0,0,0,0,0
60 SYMBOL 73,7,15,1,3,7,3,0,0
65 SYMBOL 74,252,252,252,252,248,224,224,128
70 PRINT:PRINT
75 PRINT"      AB"
80 PRINT"      CDE"
85 PRINT"      FGH"
90 PRINT"      IJ"
95 GOTO 95
```

Exercice 6.9.

Faire BORDER 0 : INK 0,26 : INK 1,0 (écriture noire, fond blanc, cadre noir).

Exercice 6.10.

Faire INK 0,11 : INK 1,15.

Exercice 6.11.

```
1 REM  Ex  6-11
2 REM
5 MODE 0
10 FOR Y=1 TO 25
20 FOR X=1 TO 20
30 P=INT(16*RND(1))
40 PAPER p
50 LOCATE X,Y:PRINT " ";
60 NEXT:NEXT
70 GOTO 70
```

Exercice 7.2.

Deux problèmes se posent : le premier est celui de l'échelle; les effectifs sont tous assez voisins (de 80 à 120), donc il faut dilater les différences, mais tout de même garder un terme constant.

Comme le numéro de la classe et l'effectif sont inscrits en tête de ligne, l'écriture commence en colonne 11. Il faut utiliser des manipulations de chaînes de caractères pour assurer une largeur constante aux impressions du numéro de classe et de l'effectif.

Nous proposons la formule de réduction d'échelle :

$$N=1/2*(C(I)-70)$$

qui fait varier N de 0 à 25.

D'où la fin du programme :

```
88 REM  Ex  7-2
89 REM
90 PRINT " CL  EFF"
100 FOR I=1 TO 10:N=INT((C(I)-70)/2)
110 PRINT RIGHT$("  "+STR$(I),3);" ";RIGHT$("  "
+STR$(C(I)),4);"  ";
120 FOR K=1 TO N:PRINT "*";:NEXT K
130 PRINT:NEXT I
```

Exercice 7.3.

```
1 REM  Ex 7-3
2 REM
10 INPUT "Centre (X,Y),Rayon";XC,YC,R
15 CLS
20 FOR T=0 TO 2*PI STEP 2*PI/100
30 X=INT(XC+R*COS(T)+0.5)
40 Y=INT(YC+R*SIN(T)+0.5)
50 LOCATE X,Y:PRINT"*"
60 NEXT
```

Attention à spécifier des valeurs qui ne font pas sortir de l'écran.

Exercice 7.4.

```
1 REM  Programme D-2B
2 REM
10 MODE 1:INK 0,0:INK 1,26:PLOT 0,200
20 FOR X=0 TO 639
30 Y=200+190*SIN(8*PI*X/640)
40 DRAW X,Y
50 NEXT
60 X$=INKEY$:IF X$="" GOTO 60
```

Exercice 7.5.

```
1 REM  Ex 7-5
2 REM
10 MODE 1:INK 1,15:R=190:CX=300:CY=200
15 PLOT CX,CY,1
20 FOR T=0 TO 2*PI STEP 2*PI/1000
30 PLOTR R*COS(T),R*SIN(T)
40 MOVE CX,CY
50 NEXT
60 GOTO 60
```

Pour avoir un disque, en 30, remplacer PLOTR par DRAWR. Bien sûr, avec un 664, on utiliserait FILL.

Exercice 7.6.

```
1 REM  Ex  7-6
2 REM
10 MODE 0
20 FOR Y=0 TO 400 STEP 2
30 FOR X=0 TO 640 STEP 4
40 P=INT(16*RND(1))
50 PLOT X,Y,P
60 NEXT:NEXT
70 GOTO 70
```

Exercice 7.7.

```
1 REM  Programme  D-3
2 REM      TELECRAN
3 REM
10 MODE 1
20 X=0:Y=399:PLOT X,Y,1
30 A=ASC(INKEY$+CHR$(0)):B=0:IF A=0 GOTO 30
35 IF A>=244 THEN B=1:A=A-4
40 IF A=243 THEN IX=1 :IY=0 :GOTO 110
50 IF A=241 THEN IX=0 :IY=-1:GOTO 110
60 IF A=242 THEN IX=-1:IY=0 :GOTO 110
70 IF A=240 THEN IX=0 :IY=1 :GOTO 110
80 IF A=16 GOTO 10
85 IF A=127 THEN PLOT X,Y,0 :GOTO 20
90 IF A=224 THEN PLOT X,Y,0 :X=320:Y=200:GOTO 15
0
95 IF A=32  THEN PLOT X,Y,0 :X=639:Y=0  :GOTO 15
0
100 GOTO 30
110 LX=(IX+1)*319.5:LY=(IY+1)*199.5:IF (X=LX) OR
 (Y=LY) GOTO 30
120 IF B=1 GOTO 140
130 PLOT X,Y,0
140 X=X+IX:Y=Y+IY
150 PLOT X,Y,1:GOTO 30
```

La ligne 110 assure qu'on ne sort pas des limites de l'écran.

Par suite de l'effet de répétition des touches, si vous voulez tracer en pointillé, vous pouvez maintenir l'appui sur la touche curseur et appuyer sur SHIFT de façon intermittente.

Nous vous suggérons d'ajouter des touches pour faire les mouvements diagonaux.

Exercice 7.8.

```
1 REM  Ex 7-8
2 REM
10 CLS
20 X=1:Y=13:D=40:LOCATE X,Y:PRINT "*":S=1
30 LOCATE X,Y:PRINT " ":X=X+1:LOCATE X,Y:PRINT "
*"
40 FOR T=1 TO D:NEXT:IF X<40 GOTO 30
50 LOCATE X,Y:PRINT " ":X=X-1:LOCATE X,Y:PRINT "
*"
60 FOR T=1 TO D:NEXT:IF X>1 GOTO 50
70 D=D+5*S:IF D=80 THEN S=-1
80 IF D=10 THEN S=1
90 GOTO 30
```

Exercice 7.9.

```
1 REM  Ex 7-9
2 REM
10 CLS
20 X=20:Y=1:D=30:LOCATE X,Y:PRINT "*"
30 LOCATE X,Y:PRINT " ":Y=Y+1:LOCATE X,Y:PRINT "
*"
40 FOR T=1 TO D:NEXT:IF Y<25 GOTO 30
50 LOCATE X,Y:PRINT " ":Y=Y-1:LOCATE X,Y:PRINT "
*"
60 FOR T=1 TO D:NEXT:IF Y>1 GOTO 50
70 GOTO 30
```

Exercice 7.10.

```
1 REM  Ex 7-10
2 REM
10 CLS
20 X=1:Y=1:D=30:LOCATE X,Y:PRINT "*"
30 LOCATE X,Y:PRINT " ":X=X+1:Y=Y+1:LOCATE X,Y:P
RINT "*"
40 FOR T=1 TO D:NEXT:IF Y<25 GOTO 30
50 LOCATE X,Y:PRINT " ":X=X-1:Y=Y-1:LOCATE X,Y:P
RINT "*"
60 FOR T=1 TO D:NEXT:IF Y>1 GOTO 50
70 GOTO 30
```

Exercice 8.1.

```
1 REM  Ex  8-1
2 REM
10 RESTORE 2000
20 FOR NN=1 TO 11
30 READ NH,ND
40 SOUND 1,NH,ND*30,10:SOUND 1,NH,10,0
50 NEXT
2000 DATA 239,1,239,1,239,1,213,1,190,2,213,2,23
9,1,190,1,213,1,213,1,239,4
```

Dans les DATA, chaque note est définie par sa hauteur et sa durée. Remarquez le 2e SOUND en 40 pour produire un court silence afin de séparer les notes.

Exercice 8.2.

SOUND 65,239,100 : SOUND 66,190,100 : SOUND 67,159,100 : RELEASE 7.

On lance une note sur chacune des voix en attente, puis on relache les 3 voix en même temps.

Exercice 8.3.

180 devient :

```
180 CLS:PRINT "Gagne":RESTORE 2000
181 FOR NN=1 TO 10:READ NH,ND:SOUND 1,NH,20*ND,1
5:SOUND 1,NH,5,0:NEXT
2000 DATA 213,1,213,2,213,1,159,2,159,1,142,2,14
2,1,106,2,127,1,159,3
```

Exercice 8.4.

```
1 REM  Jeu 'SIMON'  Ex 8-4
2 REM
10 DATA 11,239,3,213,1,190,12,179,2,159,6,142,7,
127
20 FOR N=1 TO 7:READ CO(N),NO(N):NEXT
30 MODE 0:PAPER 0:CLS
40 FOR I=1 TO 4
50 NS(I)=INT(1+7*RND(1)):NEXT
60 FOR I=1 TO 4
70 N=NS(I):SOUND 1,NO(N),50,15
80 PAPER CO(N):CLS
90 IF SQ(1)>=128 GOTO 90
100 SOUND 1,NO(N),10,0 :NEXT
110 T=0:PAPER 0:CLS
```

```
120 FOR I=1 TO 4
130 A$=INKEY$:IF A$="" GOTO 130
140 N=ASC(A$)-48:SN(I)=N
150 SOUND 1,NO(N),50,15:PAPER CO(N):CLS
160 IF SQ(1)>=128 GOTO 160
170 SOUND 1,NO(N),10,0 :NEXT
180 BIEN=-1:PAPER 0:CLS
190 FOR I=1 TO 4
200 BIEN=BIEN AND (NS(I)=SN(I)):NEXT
210 IF BIEN THEN PRINT "Gagne":GOTO 240
220 IF T=0 THEN PRINT "Reessayez":T=1:GOTO 120
230 PRINT "Perdu. C'etait ";
235 FOR I=1 TO 4:PRINT NS(I);:NEXT:PRINT
240 INPUT "On recommence ";A$
250 IF A$="oui" GOTO 30
```

NS est la séquence préparée par l'ordinateur, tandis que SN est celle proposée par le joueur. Remarquez la variable logique BIEN et sa gestion. Bien entendu, vous pouvez apporter des variantes : nombre de notes par séquence et nombre de tentatives variables, possibilité de réentendre la séquence, traitement du cas où le joueur tape sur une autre touche que 1 à 7. On a établi la correspondance :

Touche :	1	2	3	4	5	6	7
Couleur :	rose	rouge	jaune	vert	turquoise	bleu	violet
Note :	do	ré	mi	fa	sol	la	si

CONSEILS DE LECTURE

Pour maîtriser le BASIC Amstrad, connaître toutes les finesses de la programmation et découvrir le système des CPC 464 et 664, P.S.I. vous propose une palette d'ouvrages utiles.

Pour maîtriser le BASIC Amstrad

☐ **Exercices en BASIC pour Amstrad** – Maurice Charbit (Éditions du P.S.I.)

Plus de 80 problèmes vous aident à mettre en pratique vos notions de BASIC Amstrad. Toutes les solutions sont commentées en annexe.

☐ **BASIC Amstrad 1 - Méthodes pratiques** – Jacques Boisgontier et Bruno Césard (Éditions du P.S.I.)
2 - Programmes (à paraître) – Jacques Boisgontier (Éditions du P.S.I.)

Pour ceux qui ont déjà pratiqué un BASIC, voici deux ouvrages de perfectionnement du BASIC Amstrad.

☐ **BASIC plus 80 routines** – Michel Martin (Éditions du P.S.I.)

Pour pousser votre Amstrad au maximum de ses capacités : 80 routines de simulation d'instructions qui n'existent pas en BASIC Amstrad.

Pour mieux programmer votre Amstrad

☐ **102 programmes pour Amstrad** – Jacques Deconchat (Éditions du P.S.I.)

Les bases de la programmation en cinq niveaux de difficulté pour apprendre le BASIC Amstrad en jouant.

☐ **Super jeux Amstrad** – Jean-François Sehan (Éditions du P.S.I.)

50 programmes commentés pour vous permettre de créer vos propres applications ludiques.

☐ **Amstrad en famille** – Jean-François Sehan (Éditions du P.S.I.)

40 utilitaires de gestion, de jeux, etc., pour vous aider à organiser efficacement vos activités à la maison.

Pour programmer en langage machine

☐ **Assembleur de l'Amstrad** – Marcel Henrot (Éditions du P.S.I.)

Une initiation à l'assembleur du Z80, avec de nombreux programmes en langage machine.

☐ **Clefs pour Amstrad** – Daniel Martin (Éditions du P.S.I.)

Mémento présentant synthétiquement le jeu d'instructions du Z80, les points d'entrée des routines système, les connecteurs et brochage, etc. Le livre de chevet du programmeur sur Amstrad.

Ouvrages conseillés par l'auteur

☐ **Systèmes à microproceseurs** – Daniel-Jean David (Editests)

Revues

☐ **L'Ordinateur individuel**

☐ **LIST**

EN CAS D'ERREUR

Tous les listings de cet ouvrage sont des originaux sortis directement de l'imprimante. Tous les programmes proposés ici ont été testés attentivement sur Amstrad CPC 464 et 664. Une erreur serait donc exceptionnelle.

Si malgré tout, l'un des programmes ne fonctionnait pas, assurez-vous d'avoir correctement recopié le listing en BASIC.

Voici quelques conseils qui vous aideront à déceler l'erreur :
- ne confondez pas Ø (zéro) et O (lettre) ;
- assurez-vous d'avoir placé correctement les "." et ";" nécessaires ;
- comptez les lignes du programme à recopier et faites attention à ne pas en oublier ;
- donnez toujours le même nom à vos variables d'un bout à l'autre du programme.

Et surtout... Armez-vous de courage !

Votre avis nous intéresse

- Pour nous permettre de faire de meilleurs livres, adressez-nous vos critiques sur le présent livre.
- Si vous souhaitez des éclaircissements techniques, écrivez-nous, nous adresserons votre demande à l'auteur qui ne manquera pas de vous répondre directement.

- Ce livre vous donne-t-il toute satisfaction?

- Y a-t-il un aspect du problème que vous auriez aimé voir abordé?

Comment avez-vous eu connaissance de ce livre?

☐ publicité
☐ catalogue
☐ boutique micro
☐ autres
☐ cadeau
☐ librairie
☐ exposition

Avez-vous déjà acquis des livres PSI?

lesquels?

qu'en pensez-vous?

Nom Prénom Age
Adresse
Profession
Centre d'intérêt

CATALOGUE GRATUIT

Vous pouvez obtenir un catalogue complet des ouvrages PSI, sur simple demande, ou en retournant cette page remplie à votre libraire, à votre boutique micro ou aux

Editions du PSI
BP 86
77402 Lagny-sur-Marne Cedex

Achevé d'imprimer
sur les presses de l'imprimerie IBP
à Rungis (Val-de-Marne 94) 686.73.54
Dépôt légal - Septembre 1985

N° d'impression: 4858
N° d'édition: 86595-269-1
N° d'ISBN: 2-86595-269-1

P.S.I.

LA DÉCOUVERTE DE L'AMSTRAD

CPC 464 ET 664/6128

Débutant en informatique ou nouvel acheteur d'un CPC 464 ou 664/6128, "La découverte de l'Amstrad" vous propose de vous initier au BASIC Amstrad et d'apprendre à programmer votre micro afin d'utiliser au mieux ses possibilités graphiques et sonores.

De nombreux exercices, dont vous trouverez la solution à la fin de l'ouvrage, vous aideront à assimiler les instructions fondamentales du BASIC Amstrad, tandis que des annexes concises et claires vous permettront de consulter rapidement les mots-clés du BASIC Amstrad, les messages d'erreur, etc.

9 782865 952694

ÉDITIONS DU P.S.I.
BP 86 - 77402 LAGNY S/MARNE CEDEX - FRANCE

ISBN 2-86595-269-X 115 FF

PHOTO : L. MAUNOURY

Document numérisé
avec amour par :

AMSTRAD
CPC

MÉMOIRE ÉCRITE

http://amstradcpc.fredisland.net/