
BASIC SANS PEINE

Auto-initiation au BASIC MO5 — TO7/70

CODICIEL

livre accompagné de 2 cassettes



**cedic
nathan**

Couverture : Walter Lalonde
Illustrations : Henri Favre et Raoul Raba
Maquette : Michèle Beaucamp et Alain Dufourcq

Ce volume porte la référence
ISBN 2-7124-4050-1

Toute reproduction, même partielle, de cet ouvrage est interdite. Une copie ou reproduction par quelque procédé que ce soit, photographie, photocopie, microfilm, bande magnétique, disque ou autre, constitue une contrefaçon passible des peines prévues par la loi du 11 mars 1957 sur la protection des droits d'auteur.

© CEDIC 1984

CEDIC, 32, boulevard Saint-Germain, 75005 - PARIS

SOMMAIRE

Prologue	9
Introduction	11
Suivi de la cassette :	
Comment utiliser les cassettes d'auto-initiation ?	12
Chargement et feuilletage des chapitres	12
La structure du logiciel d'auto-initiation	14
La page "DÉFINITION"	15
La page "SYNTAXE"	16
La page "APPLICATION"	16
La page "EXERCICE"	17
Compléments, activités et expériences :	
La programmation d'un micro-ordinateur	19
Vos premiers ordres en BASIC : LOAD et RUN	19
Avoir la main ou ne pas l'avoir ?	21
La touche ENTRÉE	23
Les instructions de programme	27
Ce que l'on voit sur l'écran... et ce qui est dans la mémoire	28
Une feuille et des tiroirs	30
CHAPITRE 1. Écriture dans l'écran	
Suivi de la cassette	35
Premières pages	35
LOCATE	36
PRINT	40
COLOR	42
SCREEN	44
Compléments, activités et expériences :	
Le curseur invisible	46
Des calculs dans LOCATE	
Les limites de l'écran (LOCATE)	47
Le défilement de l'écran (PRINT)	48
Mettre ou ne pas mettre... des guillemets (PRINT)	48
Couleurs de fond et d'écriture (COLOR)	48
Quelques astuces sur la page d'écriture (SCREEN)	49
L'encre sympathique (COLOR, SCREEN)	50
Les paramètres sous entendus	51

CHAPITRE 2. Graphique et son

Suivi de la cassette	53
LINE	54
BOX BOXF	56
PSET	59
Couleur	60
PLAY	61
Compléments, activités et expériences :	
Des cadres assortis	64
Respecter les multiples de 8	65
Les couleurs en cours	66
Effets de cartes	67
Des boîtes en couleur de fond	67
Les “bavures” de couleurs	68

CHAPITRE 3. Données et variables

Suivi de la cassette	71
Les données	72
Données numériques	73
Données alphanumériques	75
Compléments, activités et expériences :	
L’affectation	77
L’échange de valeurs	78
Petits et grands nombres	78
Les bons et les mauvais caractères	79
Serrer les affichages	80

CHAPITRE 4. Les opérations

Suivi de la cassette	81
Opérations numériques	81
Opération alphanumérique	83
Opérations logiques	84
Compléments, activités et expériences :	
La prise de puissance	86
La pratique du module	86
Des chiffres et des nombres	88
La priorité des opérations	89
Les comparaisons enchaînées	90

CHAPITRE 5. Notions de programmation

Suivi de la cassette	91
Modes	91
Ligne	92
RUN END	94
STOP CONT	95

CHAPITRE 6. Aides à la programmation

Suivi de la cassette	97
LIST	97
DELETE	98
REM	98
TRON TROFF	99
Compléments, activités et expériences :	
Commentez vos programmes	100
Les faux commentaires	101
Suivre à la trace	102

CHAPITRE 7. Caractères utilisateurs

Suivi de la cassette	103
CLEAR	103
DEF GR\$	104
GR\$	106
Compléments, activités et expériences :	
Des lettres grecques, des animaux, des symboles	109
Qui rit vendredi, dimanche pleurera	111
Un dessin animé	112
Décodage d'une forme	112

CHAPITRE 8. Entrée des données

Suivi de la cassette	115
INPUT	116
INKEY\$	119
DATA READ	120
RESTORE	122
Compléments, activités et expériences :	
Input or not input ?	123

Les DATA	124
Le bouclage sur les entrées	125
La guerre des étoiles	125

CHAPITRE 9. Branchements

Suivi de la cassette	127
GOTO	127
IF THEN ELSE	128
ON GOTO	129
Compléments, activités et expériences :	
Le piège à conditions	131
La boucle ...TANT QUE...	132
Aiguillages	135
Plus, moins ou égal ?	136

CHAPITRE 10. Instructions répétitives

Suivi de la cassette	137
FOR NEXT	137
GOSUB RETURN	140
Compléments, activités et expériences :	
Ifoufor ?	142
Quelques cas "limites"	142
Quelques effets graphiques	143
Le travail en équipe	144
L'écriture descendante	146

CHAPITRE 11. Tableaux

Suivi de la cassette	149
DIM	150
A(i) A\$(j)	152
Compléments, activités et expériences :	
Somme, tri,...	154
Lecture de "data"	156
Affectation de tableaux	157
Tracé d'un polygone	158

CHAPITRE 12. Fonctions numériques

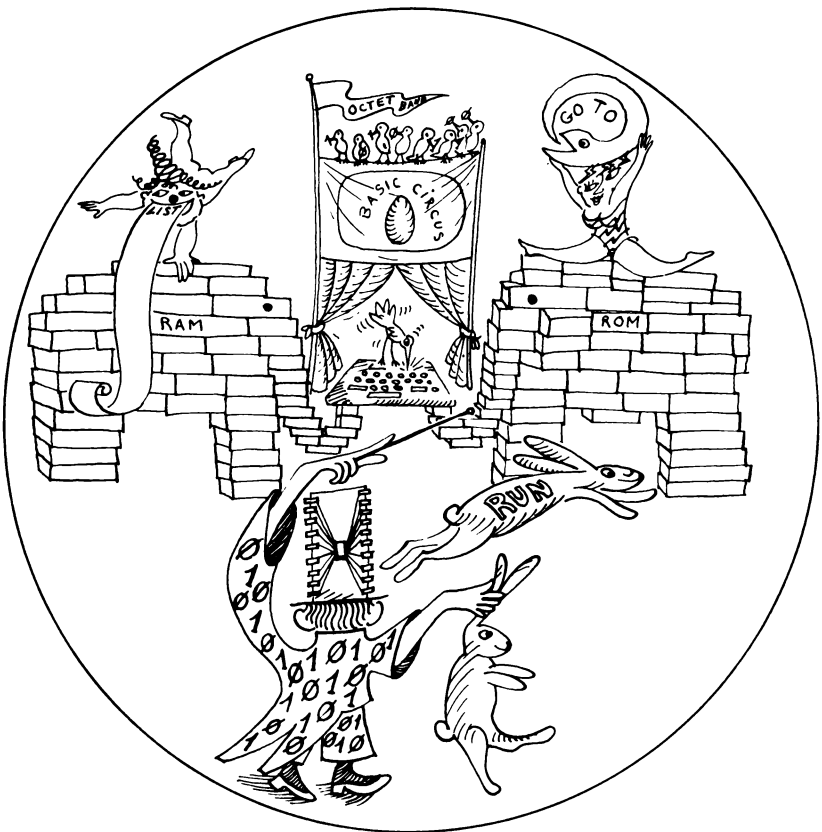
Suivi de la cassette	159
INT	159
SGN	160
RND	161
Autres fonctions	163
Compléments, activités et expériences :	
Un nombre est-il entier ?	164
Les chiffres d'un nombre	164
Choisir au hasard certaines listes de données	165
Des cercles	166
Des polygones	167
Tracé de courbes	168

CHAPITRE 13. Fonctions alphanumériques

Suivi de la cassette	171
LEFT\$ RIGHT\$ MID\$	172
VAL STR\$	173
LEN	174
ASC CHR\$	174
INSTR	175
Compléments, activités et expériences :	
Le verlan	178
Détruire et construire des mots	178
Les caractères cachés	180
Afficher des chiffres serrés	180
Entrée sans INPUT	181

CHAPITRE 14. Disposition des résultats

Suivi de la cassette	183
TAB SPC	183
PRINTUSING	184
ATTRB	185
Compléments, activités et expériences :	
Tables et tableaux	196
Choisir ses caractères	187
Créer vos cercles "pleins"	188
Annexe	190



Quelle différence y a-t-il entre un bon programmeur et un bon prestidigitateur ?

Aucune, bien sûr ! Car chacun a pour objectif de présenter, sur un écran ou une scène, un spectacle agréable et bien huilé, mais dont les rouages restent cachés aux spectateurs ou à l'utilisateur. Tout l'art consiste à faire sortir au bon moment les foulards du chapeau, les dessins sur l'écran, les colombes d'une manche ou les bonnes suites de caractères.

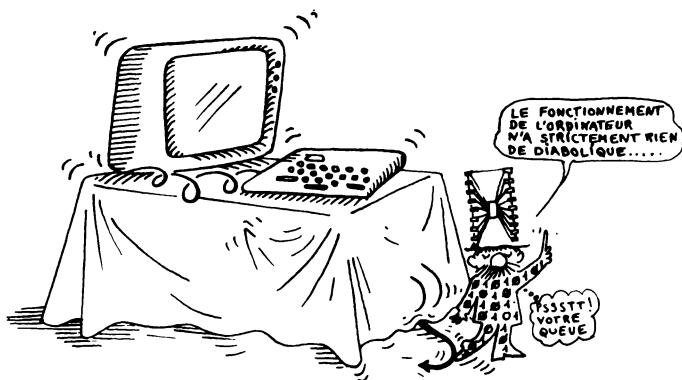
Chacun, aussi, a passé des heures et des heures à préparer l'exécution de son numéro, de manière que, lorsque le rideau s'ouvre les actions et les jeux s'enchaînent les uns aux autres selon le scénario, l'algorithme, le programme parfaitement imaginé, écrit et répété à l'avance.

Mais quelle récompense de voir tout se passer comme prévu, sans qu'apparaissent, alors, la difficulté, les efforts, le travail de la mise au point préalable ! C'est cela la "magie" ! Et vous ne résisterez sûrement pas à l'ensorcellement qui saisit l'apprenti programmeur que vous allez devenir. Car le but de ce livret et des deux cassettes qui l'accompagnent est de faire de vous un magicien du BASIC. Votre initiation doit maintenant commencer, c'est à vous de lire et d'agir...

INTRODUCTION

Les cassettes d'auto-initiation au BASIC MO5 ou TO7(-70) et ce livre sont complémentaires.

Les deux cassettes vous apprendront la signification des principales instructions du BASIC ; les applications simples et les exercices qu'elles vous proposent vous aideront à comprendre et à mémoriser ce langage.



Chapitre par chapitre, ce livret d'accompagnement apporte, d'une part, les précisions et compléments nécessaires et, d'autre part, il propose des activités, programmes et expériences à réaliser indépendamment des cassettes.

Une méthode de travail efficace consiste donc à :

— Lire la suite de cette introduction :

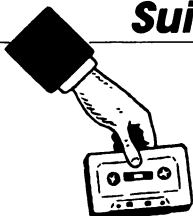
- Comment utiliser les cassettes d'auto-initiation
- Quelques précisions utiles sur la programmation d'un micro-ordinateur

— Pour chaque chapitre (du numéro 1 au numéro 14) :

- Travailler d'abord avec la cassette en consultant la partie correspondante du livret. ("suivi de la cassette")
- Finir de lire le chapitre du livret en faisant les activités proposées. ("Compléments, activités et expériences")

COMMENT UTILISER LES CASSETTES D'AUTO-INITIATION ?

Suivi de la cassette



Chargement et feuilletage des chapitres

La cassette 1 contient les chapitres 1 à 6.

La cassette 2 contient les chapitres 7 à 14.

Pour charger une cassette :

0. Brancher et mettre sous-tension l'écran, le lecteur et le MO5.

Si vous avez un TO7(-70), frappez sur **1** à l'affichage de la première page.

1. Placer la cassette dans le lecteur.

2. Rembobiner la cassette.

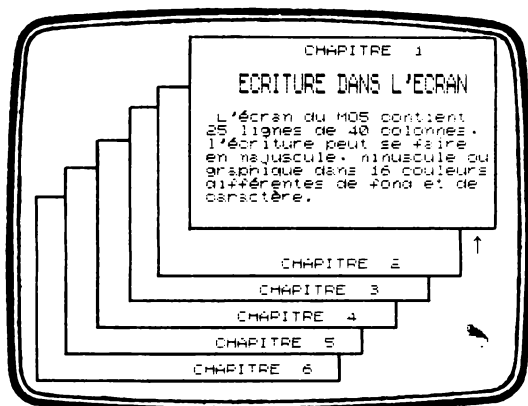
3. Appuyer sur la touche "lecture" du lecteur.

4. Frapper :

R U N " ENTREE

La page "titre" apparaît d'abord.

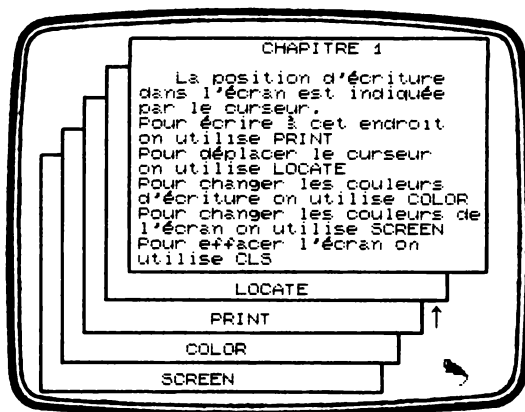
Puis après 2 ou 3 minutes apparaissent les différentes pages-têtes de chapitre.



Une petite flèche indique le chapitre que vous pouvez choisir en frappant sur **ENTREE** .

Pour déplacer la petite flèche, utilisez les touches **◀** et **▶** .
Pour choisir le chapitre pointé par la petite flèche, frappez **ENTREE** .

Après l'appui sur **ENTREE** , un autre ensemble de pages est affiché ; la première page de cet ensemble est toujours écrite complètement.



Vous pouvez donc ainsi “feuilleter” le contenu des chapitres de plus en plus profondément.

Mais il se peut que vous vouliez revenir en arrière

Pour revenir à l'ensemble de pages précédent, frappez la touche **RAZ** .

Vous pouvez aussi vouloir avancer plus vite ou bien vouloir accéder à la première page de l'ensemble qui suit.

Pour passer à l'ensemble de pages suivant, frappez la touche **▶** .

Enfin, sachez que la touche **ACC** vous permet de passer d'un seul coup à la page la plus en avant déjà étudiée préalablement.

Vous connaissez maintenant toutes les touches utiles au “feuilletage” de votre logiciel d'auto-initiation.

Nous vous conseillons de faire quelques essais avant de commencer l'étude proprement dite.

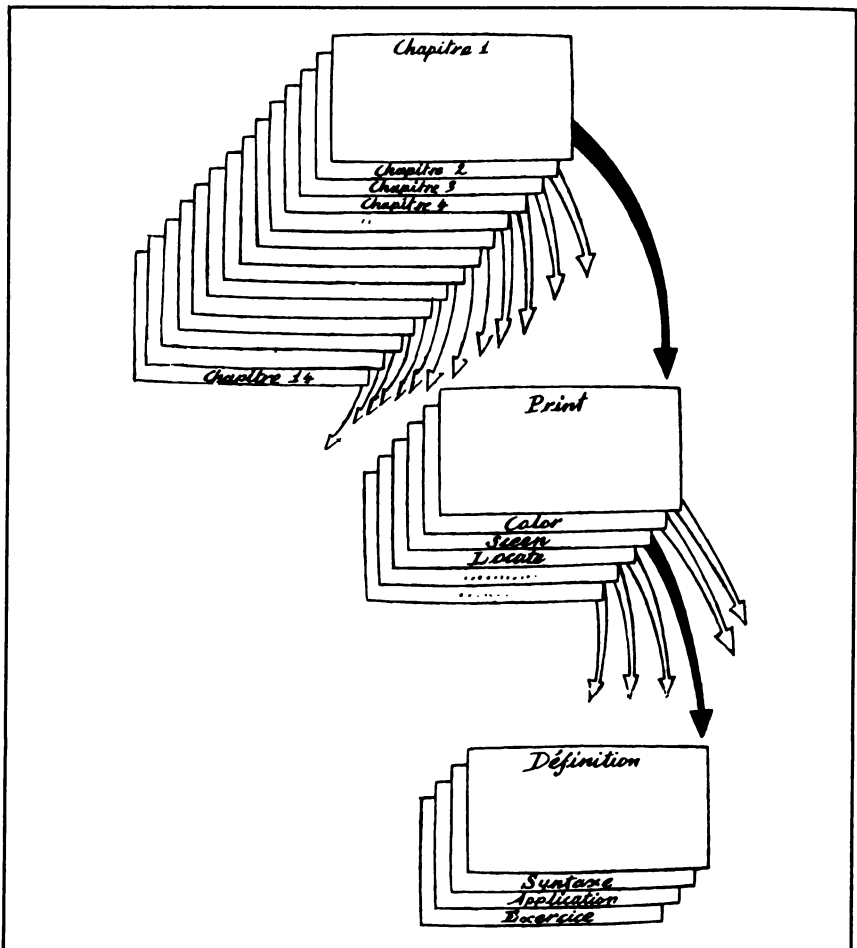
La structure du logiciel d'auto-initiation

Le logiciel est divisé en 14 chapitres.

Chaque chapitre est divisé en 3, 4 ou 5 “paragraphes” consacré chacun à une instruction. Par exemple, le chapitre 1 est divisé en 4 paragraphes : PRINT, COLOR, SCREEN, LOCATE.

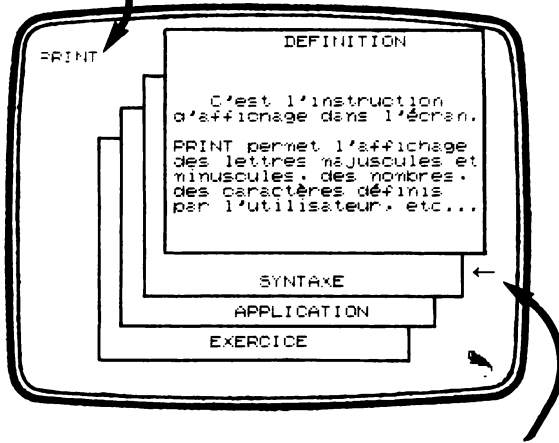
Chaque paragraphe est divisé en 4 “pages” nommées :

- Définition
- Syntaxe
- Application
- Exercice



Ce que vous voyez sur l'écran :

Titre de la page affichée.

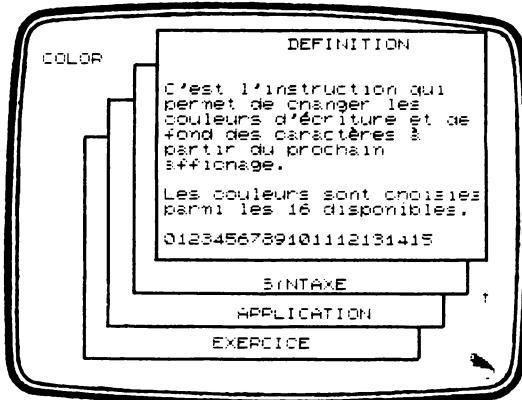


La petite flèche pointe sur la page affichable lorsque vous appuyez sur **ENTRÉE**.
Vous pouvez aussi appuyer sur **RAZ** ou **↵**.

La page "DEFINITION"

Un texte très court définit le mot ou l'instruction dont le nom est écrit en haut à gauche.

Exemple

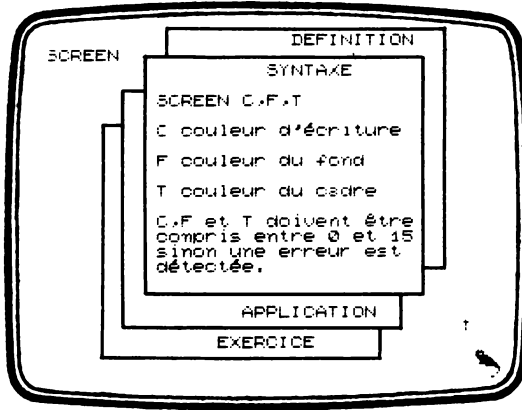


La page "SYNTAXE"

Cette page indique la manière d'écrire correctement des instructions qui utilisent le mot écrit en haut à gauche.

Des compléments sont donnés dans le livret.

Exemple



Attention

Si les explications nécessitent plus d'une page, la petite flèche reste sur la page syntaxe après son affichage (et elle devient rouge).

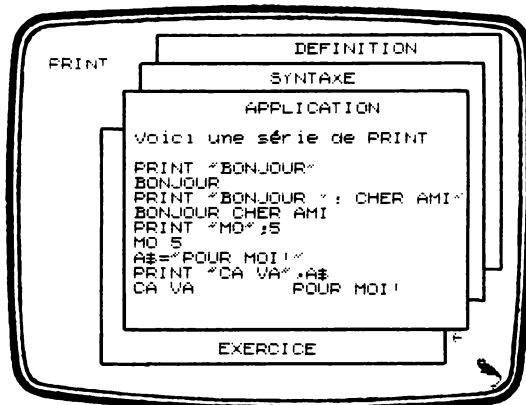
En appuyant sur **ENTREE** vous consulterez cette deuxième page.

La page "APPLICATION"

Pendant le déroulement de cette page, vous n'avez rien d'autre à faire que regarder et comprendre.

Le MO5 ou le TO7(-70) vous présente ou bien une seule instruction, ou bien un programme (comme si vous le tapiez caractère par caractère), puis il le fait exécuter (comme si vous frappiez **R U N**). Chaque instruction est alors exécutée l'une après l'autre : le numéro de l'instruction exécutée se change en rouge et l'action qu'elle provoque apparaît sur l'écran.

Exemple



Attention

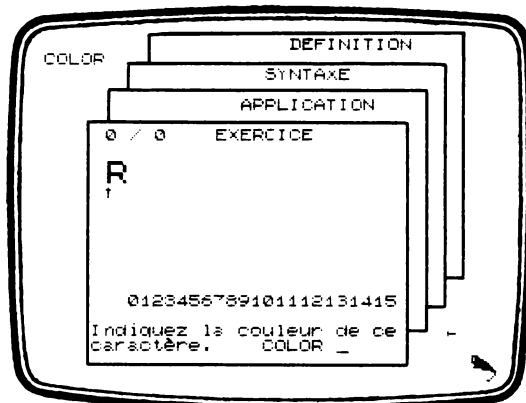
Si vous voulez étudier plus à loisir ce qui se passe sur l'écran, vous pouvez appuyer sur la touche **STOP**. Toute exécution est alors interrompue et vous pouvez examiner en détail tout ce qui est alors affiché à l'écran.

Pour continuer l'exécution, il suffit d'appuyer sur une touche quelconque, par exemple sur la touche **■**.

La page "EXERCICE"

Un exercice vous est proposé pour chaque instruction, afin d'en assimiler et d'en mémoriser les éléments principaux.

Exemple



LA PROGRAMMATION D'UN MICRO-ORDINATEUR

Compléments, activités et expériences



Vos deux premiers ordres en BASIC :

LOAD et RUN

Sur le MO5
lorsque le message :
© BASIC Microsoft 1984
OK
est affiché sur votre écran...

Sur le TO7(-70)
lorsque vous avez
appuyé sur **F1**
à l'affichage
de la première page...

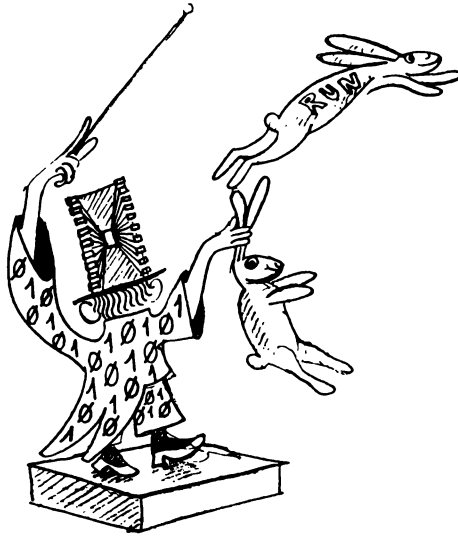
vous pouvez "parler BASIC" avec votre micro-ordinateur.

Si vous ne parlez pas BASIC, vous pouvez alors confier l'organisation de votre dialogue à un programme écrit par quelqu'un d'autre et enregistré sur cassette. Ce programme vous proposera de vous montrer des images et de vous faire agir sur ces images grâce à des "mots" simples frappés au clavier. Par exemple, les 2 cassettes d'auto-initiation vous permettront de commander votre apprentissage en frappant sur des touches comme **RAZ** **↶** **■** **■** ou sur des numéros ou encore sur certaines suites bien précises de lettres qui vous seront indiquées au fur et à mesure de votre avance.

Lorsque les programmes figurant sur ces cassettes ont été chargés dans votre micro-ordinateur, vous n'avez donc plus à vous soucier du langage de programmation compris par la machine puisque le programme sur cassette joue le rôle d'une sorte d'interprète. En définitive, le seul ordre-BASIC qu'il vous faut absolument connaître est celui qui commande le chargement d'un programme à partir d'une cassette vers la mémoire du micro-ordinateur. Cet ordre s'énonce sur le clavier :

L O A D **ENTREE**

Cependant cet ordre ne suffit pas tout à fait ; en effet, lorsqu'un programme se trouve dans la mémoire d'un ordinateur, on ne le voit pas et le fait qu'il soit là ne sert pas à grand-chose. Ce qui est intéressant c'est de le faire exécuter.



L'ordre qui commande l'exécution d'un programme en mémoire s'énonce ainsi sur le clavier :

R U N ENTREE

La suite des opérations vous échappe alors puisqu'elle ne dépend que des instructions figurant dans le programme en mémoire ; vous ne pouvez alors intervenir que si ces instructions prévoient que la machine s'arrête, soit pour vous poser une question, soit pour vous laisser choisir telle ou telle suite possible.

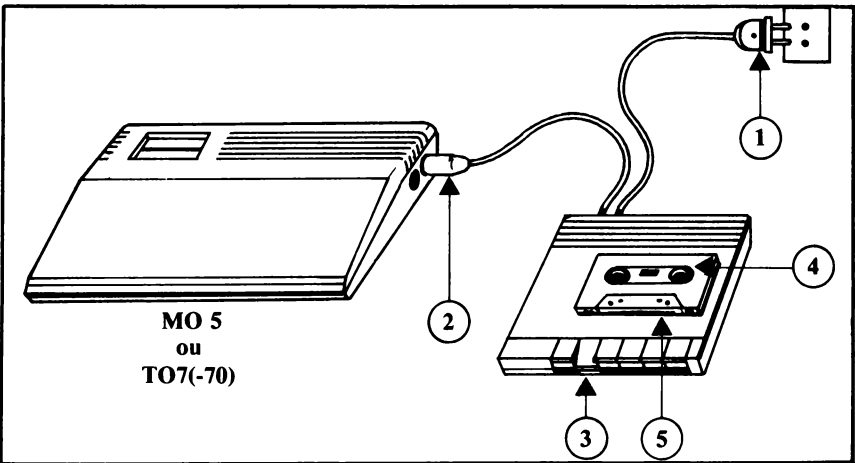
En fait, il existe un ordre qui équivaut à la succession des deux ordres précédents : il commande à la machine de charger en mémoire le programme sur cassette puis de l'exécuter. Cet ordre s'énonce ainsi sur le clavier :

R U N " ENTREE

Attention

Lorsque vous donnez l'ordre LOAD, l'ordinateur renvoie au lecteur de cassette l'ordre de lire ce qui est enregistré sur une cassette. Il vous faut donc vérifier que :

1. Le lecteur de cassette est bien sous-tension.
2. Le lecteur de cassette est bien relié au micro-ordinateur par son cordon.
3. La touche "lecture" du lecteur de cassette est bien enfoncée.
4. Une cassette se trouve bien dans le lecteur.
5. Le programme que vous désirez charger est bien celui qui se trouve juste après la tête de lecture du lecteur. (Si cela n'est pas le cas, il vous faut peut-être rembobiner la bande ou bien consulter le compteur et faire une "avance rapide" de bande...).



Avoir la main, ou ne pas l'avoir

Lorsque le message "OK" est affiché à l'écran, la balle est donc dans votre camp ; la machine attend vos ordres ; en quatre mots : *vous avez la main*.

Si vous commandez alors RUN, le contrôle des opérations ne vous appartient plus : *vous perdez la main*. Cela se traduit par le fait que tout ce que vous frappez sur le clavier n'a aucun effet (à moins évidemment que le programme ne vous ait repassé la main sans que vous vous en soyez aperçu).

Ne vous en faites pas !

Vous pourrez toujours reprendre la main d'au moins deux manières :

La première est un peu brutale et consiste à appuyer sur *la touche "initialisation"* située en haut à gauche du clavier ; il se produit alors un flash, l'écran se vide et le message sécurisant "OK" apparaît en haut de l'écran. (Sur le TO7-70, cette touche est en haut, à droite.)

La deuxième est plus "magique" car vous n'auriez vraiment pas pu l'inventer tout seul : il vous faut appuyer simultanément sur les 2 touches **CNT** et **C**. Le message "Break in... - OK" apparaît alors quelque part sur l'écran.

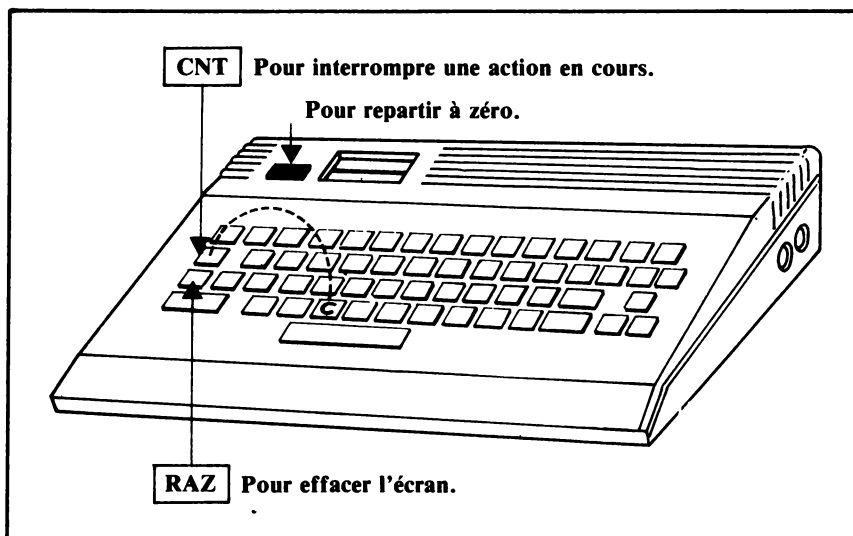
Dans les deux cas vous pouvez alors frapper ce que vous désirez et reparler BASIC avec la machine. Par exemple, vous pouvez donner l'ordre RUN et l'exécution du programme en mémoire est relancée à son début.



Attention

Si vous avez la main et si l'écran est encombré de dessins ou d'écritures, il se peut que vous écriviez un ordre sur une ligne où figurent aussi des gribouillis qui troublent l'énoncé de votre ordre. Suivez alors ce conseil :

Frappez sur la touche **RAZ**, avant de donner un ordre.



La touche entrée

Le clavier d'un micro-ordinateur ressemble au clavier d'une machine à écrire. Parmi les quelques différences visibles à l'œil nu, la plus importante est la touche **ENTREE**.

C'est vraiment la touche qui fait la différence ! Tant que vous frappez sur une autre touche, vous pouvez suivre sur l'écran la trace exacte de ce que vous faites. Mais lorsque vous frappez sur **ENTREE**, il se passe vraiment des choses tout à fait extraordinaires et que vous devez connaître.

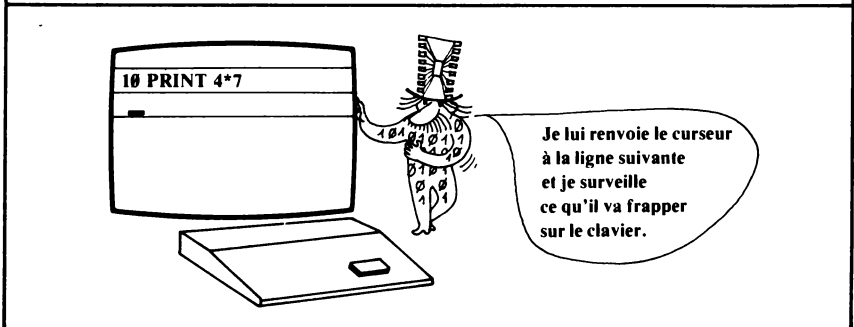
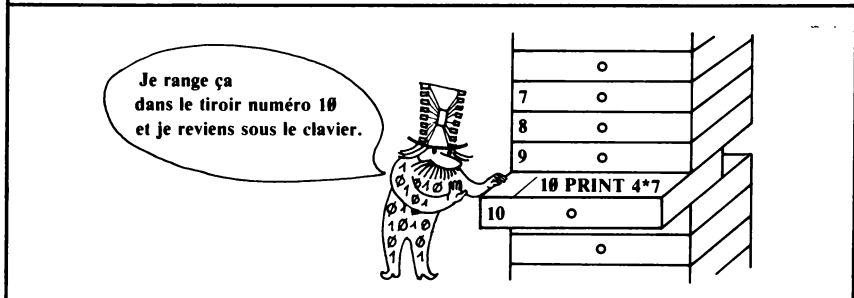
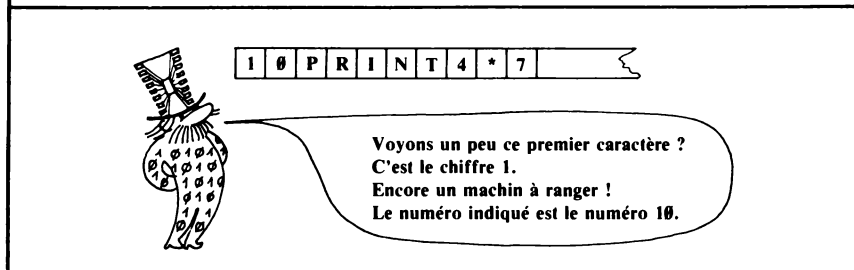
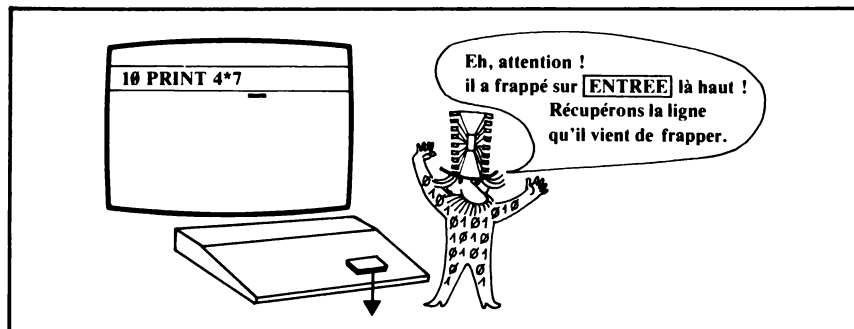
On peut se risquer à le dire : si vous comprenez bien le rôle de cette touche, alors vous aurez réussi votre "entrée" dans le monde du BASIC.



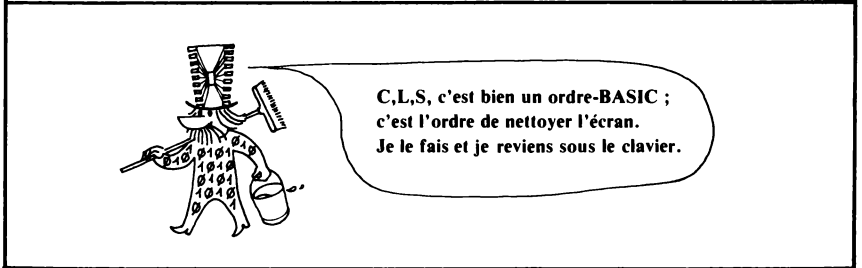
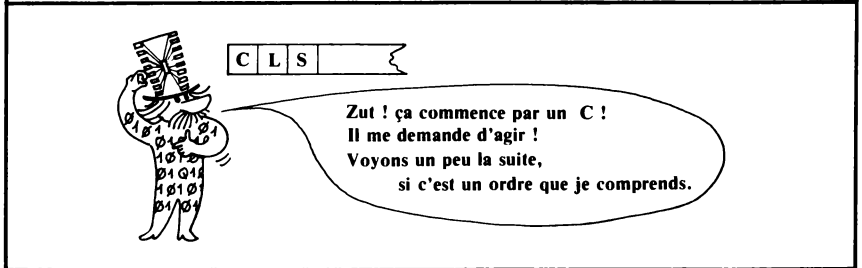
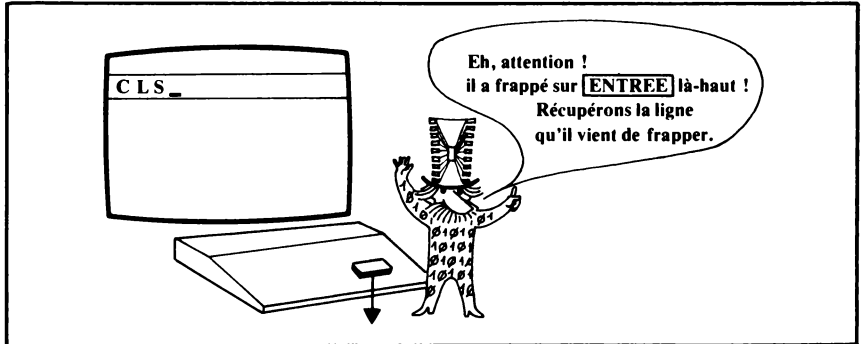
Que se passe-t-il donc à l'intérieur du micro-ordinateur lorsque vous frappez sur la touche **ENTREE** ?

L'interpréteur qui se cache sous le clavier se met en action : il "comprend" que l'utilisateur frappeur de touches vient de terminer une ligne et qu'il s'agit maintenant de décider ce que l'on fait de la suite des caractères de cette ligne. Ce qui va déterminer la nature des choses à faire dépend essentiellement du tout premier caractère de la ligne. Il y a deux cas possibles :

— Si le premier caractère de la ligne frappée est un chiffre, la chose à faire n'est qu'un simple rangement : l'interpréteur range alors la suite de caractères frappés (qui commence par un numéro) dans sa mémoire et renvoie le curseur à la ligne suivante de l'écran dans l'attente des frappes suivantes.



— Si le premier caractère de la ligne frappée n'est pas un chiffre, la chose à faire est indiquée par la suite des caractères ; vous avez alors énoncé un ordre en BASIC et cet ordre doit être immédiatement exécuté.





Attention

Si la suite de caractères que vous avez frappée avant **ENTREE** n'est pas un ordre BASIC reconnaissable par l'interpréteur, alors un message d'erreur sera affiché sur l'écran.

Exemple : la frappe de :

BONJOUR **ENTREE**

provoque une erreur de syntaxe signalée par le message :

ERROR 2

sur le MO5

SN ERROR

sur le TO7(-70)

“Bonjour” n'est pas un ordre-BASIC !



Les instructions de programme

Les lignes commençant par un numéro sont des “instructions de programme”.

A quoi peut bien servir d'écrire des lignes commençant par un numéro si la seule chose qu'en fait la machine consiste à les ranger consciencieusement dans sa mémoire ?

S'il ne s'agissait de ranger qu'une ligne, cela serait un peu bête ; mais après avoir ainsi mis la ligne numéro 1 en mémoire, vous pouvez frapper et mettre en mémoire la ligne numéro 2 ; puis la ligne numéro 3,... et ainsi de suite, vous pourrez ranger des dizaines et même des centaines de lignes.

La suite de ces lignes en mémoire constitue alors un programme pouvant décrire des actions très complexes à accomplir. Une ligne de programme s'appelle *une instruction*. L'avantage du rangement en mémoire est alors le suivant : à tout moment vous pouvez donner l'ordre d'exécuter cette suite d'instructions en mémoire et le micro-ordinateur fera alors tout ce qui est indiqué dans ces instructions à une vitesse fabuleuse (quelques centaines d'instructions par seconde). Et vous aurez alors le plaisir de voir exécuter devant vous la suite de tous les ordres que vous aurez énoncés dans chaque ligne. Vous connaissez déjà l'ordre d'exécution de la suite des instructions en mémoire : c'est l'ordre RUN !

Finalement vous savez maintenant ce qu'il faut faire pour écrire et faire exécuter un programme-BASIC par votre micro-ordinateur. Voici un exemple qui résume les 3 points importants :

Exemple

1. Vérifiez que vous avez la main en appuyant, par exemple, sur **RAZ** .

L'écran s'efface, vous pouvez commencer !

2. Frappez un programme, c'est à dire une suite d'instructions numérotées, par exemple :

1 A = 7 ENTREE

2 B = 3 2 ENTREE

3 C = A * B ENTREE

4 ? C ENTREE

Vous voyez que la machine n'a fait aucune opération. Elle attend un ordre.

3. Ordonnez l'exécution en frappant :

R U N ENTREE

La machine exécute une à une chacune des instructions du programme en mémoire et vous voyez s'afficher le nombre :

224

Ce qui s'est exactement passé est expliqué dans la fin de ce chapitre.



Attention

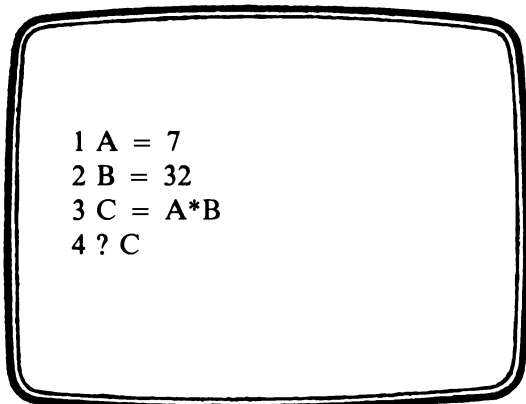
Lorsque l'exécution du programme est terminée, ce programme est toujours en mémoire. Si vous voulez frapper un autre programme, vous avez donc intérêt à effacer d'abord celui qui s'y trouve déjà. L'ordre d'effacement de la mémoire s'énonce en BASIC :

N E W ENTREE

Ce que l'on voit sur l'écran...

... et ce qui est dans la mémoire

Lorsque vous frappez la suite des instructions d'un programme, elles s'inscrivent au fur et à mesure sur l'écran. Dans le cas du paragraphe précédent vous avez pu voir affiché :



(La frappe sur la touche **ENTREE** ne laisse pas de trace : le curseur passe simplement à la ligne.)

Dans cette situation, si vous frappez sur **RAZ**, l'écran est effacé complètement.

Votre programme est-il alors perdu ?

Non ! car le programme avait été rangé en mémoire instruction par instruction. L'écran seul a été effacé, mais le programme est encore présent dans la mémoire.

Comment voir sur l'écran la suite des instructions présentes dans la mémoire ?

Il existe un ordre-BASIC qui fait afficher à l'écran la liste des instructions en mémoire. Cet ordre s'énonce :

L I S T ENTREE

En frappant cela vous verrez donc apparaître votre programme sur l'écran.

Frappez alors :

N E W ENTREE

Rien n'a été effacé sur l'écran !

La liste du programme y est toujours affichée. Et pourtant, le programme a été effacé de la mémoire !

Vous pourrez d'ailleurs le constater vous même en frappant successivement :

RAZ

L'écran s'efface alors. Tout est-il perdu ?

Le programme est-il encore en mémoire ?

L I S T ENTREE

... et rien ne s'affiche ; ce qui prouve qu'aucune instruction n'est actuellement en mémoire !

Une feuille et des tiroirs...

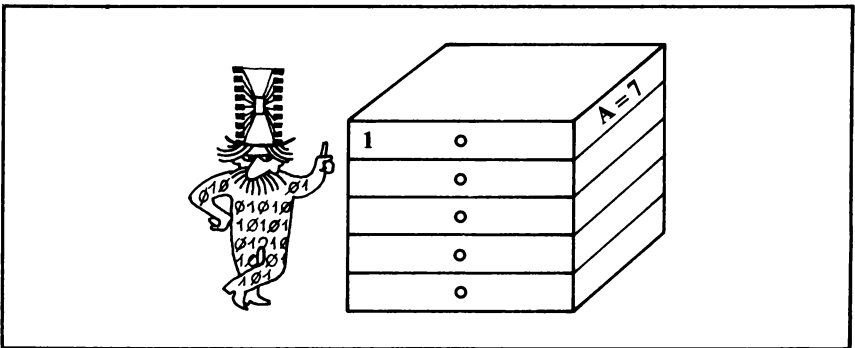
Finalement, on pourrait prétendre que le travail d'un micro-ordinateur se résume à quelques tâches bien simples :

- Ranger dans des tiroirs (ses mémoires).
- Recopier sur une feuille (l'écran).
- De temps en temps faire une opération.

Détaillons, par exemple, les actions déclenchées dans la machine par l'écriture et l'exécution du petit programme écrit plus haut.

1 A = 7 ENTREE

L'appui sur **ENTREE** provoque le rangement de l'instruction numéro 1 dans le "tiroir" numéro 1.



2 B = 3 2 ENTREE

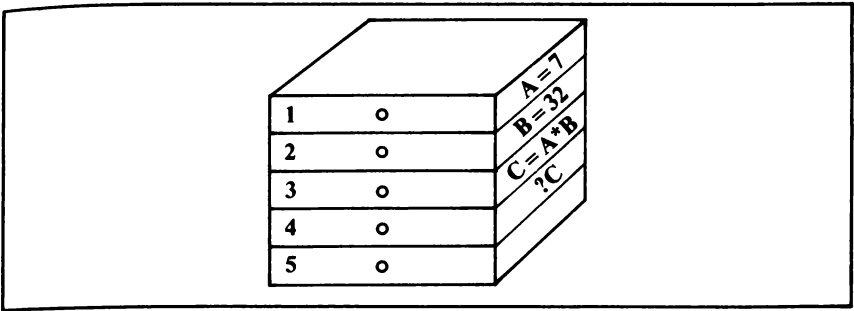
L'appui sur **ENTREE** provoque le rangement de l'instruction numéro 2 dans le tiroir numéro 2.

3 C = A * B ENTREE

L'appui sur **ENTREE** provoque le rangement de l'instruction numéro 3 dans le tiroir numéro 3.

4 ? C

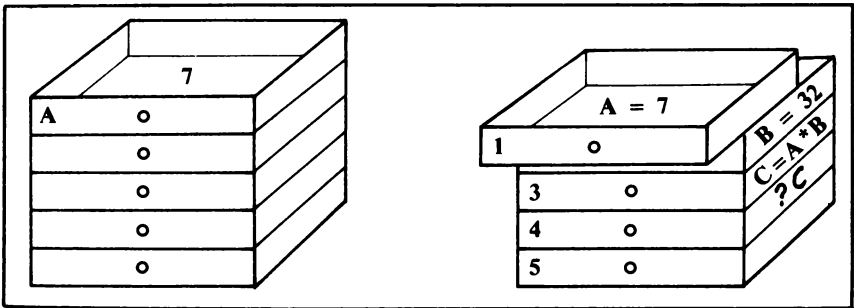
L'appui sur **ENTREE** provoque le rangement de l'instruction numéro 4 dans le tiroir numéro 4.



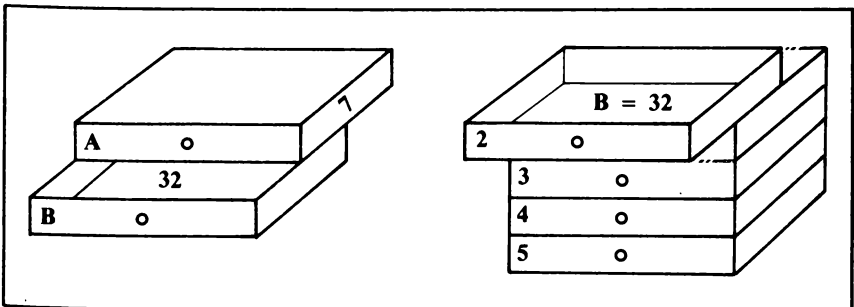
R U N ENTREE

L'appui sur **ENTREE** provoque l'interprétation de l'ordre donné (RUN) et donc, l'exécution de l'instruction du tiroir numéro 1.

Cette instruction ($A = 7$) provoque d'une part l'ouverture et l'étiquetage d'un tiroir au nom de "A", d'autre part, le rangement du nombre 7 dans ce tiroir.



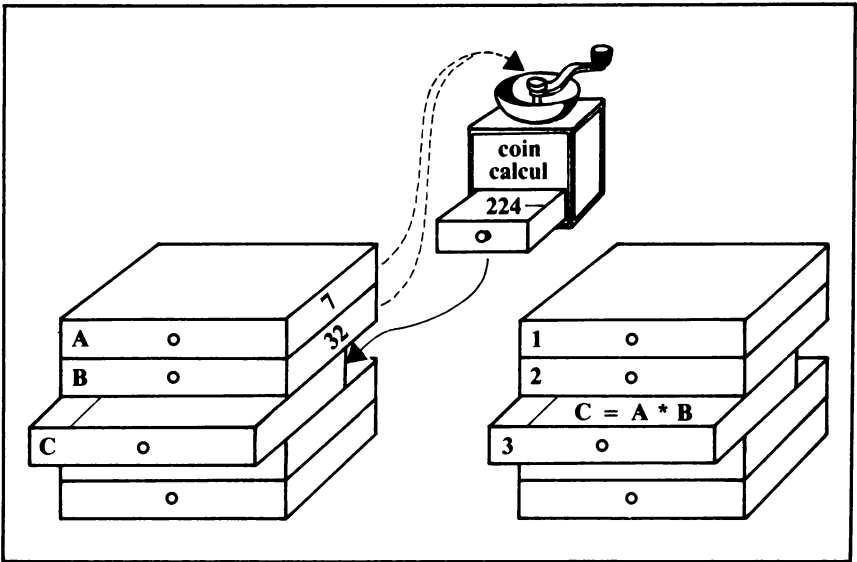
L'instruction numéro 1 étant exécutée, la machine exécute l'instruction suivante (numéro 2). Celle-ci ($B = 32$) provoque le rangement du nombre 32 dans un nouveau tiroir étiqueté "B".



L'instruction numéro 2 étant exécutée, la machine exécute l'instruction suivante (numéro 3) :

$$C = A * B$$

D'une part, la machine ouvre un nouveau tiroir étiqueté "C"; mais ce qu'elle va mettre dedans lui demande un petit travail préalable : le calcul du nombre "A * B". Pour cela, elle recopie quelque part le contenu du tiroir A et le contenu du tiroir B, elle amène ces deux copies dans son "coin-calcul" et effectue l'opération indiquée (ici la multiplication). Le résultat (ici 224) est donc rangé dans le tiroir étiqueté "C".



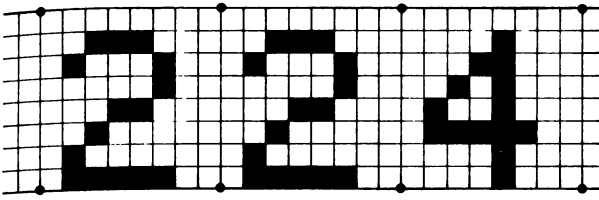
L'exécution de l'instruction suivante, numéro 4:

4 PRINT C

(le **2** équivaut aux cinq lettres **P R I N T**)

provoque la "recopie" du contenu du tiroir étiqueté "C" sur l'écran.

Évidemment cette recopie n'est pas une simple reproduction de ce que contient le tiroir "C". Il y a décodage du contenu de "C" (attention, C est un tiroir électronique) puis recodage pour que nos yeux voient sur l'écran, ce que notre cerveau interprète comme le nombre "deux cent vingt-quatre".



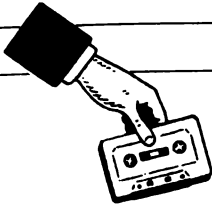
Mais, ceci est une autre histoire...

Avertissement :

La première partie de chaque chapitre, intitulée “Suivi de la cassette”, est un complément à l’étude du logiciel sur cassettes. Il n’est donc pas conseillé de lire cette partie sans le contexte du logiciel.

CHAPITRE 1

ECRITURE DANS L'ECRAN



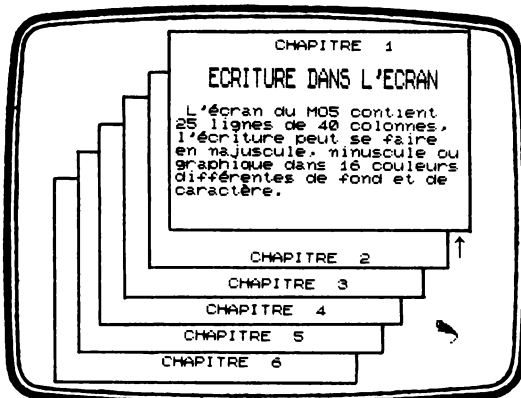
Suivi de la cassette

Ce premier chapitre très simple va surtout vous permettre de vous familiariser avec les touches du clavier et la manière de tourner les "pages" du logiciel d'auto-initiation.

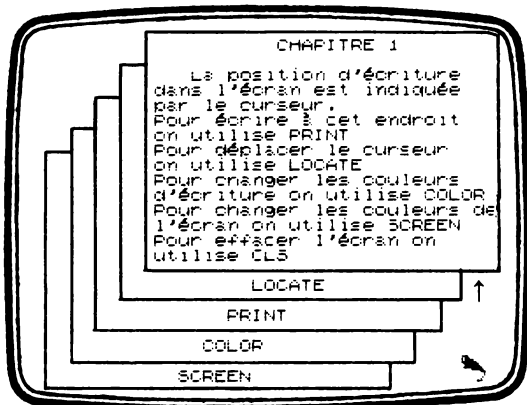
Premières pages

Les premières instructions que vous allez étudier, tout en vous habituant au maniement du clavier et du logiciel, sont très simples : ce sont les instructions permettant d'afficher des textes ou des résultats de calculs sur votre écran et à l'endroit que vous voulez sur cet écran.

Le premier écran qui apparaît est le suivant :



Frappez sur **ENTREE** pour “pénétrer” dans le chapitre 1.



La première page de chaque chapitre est toujours une page de discours de présentation. Elle vous offre le choix entre plusieurs instructions à étudier ; ici LOCATE, PRINT, COLOR ou SCREEN.

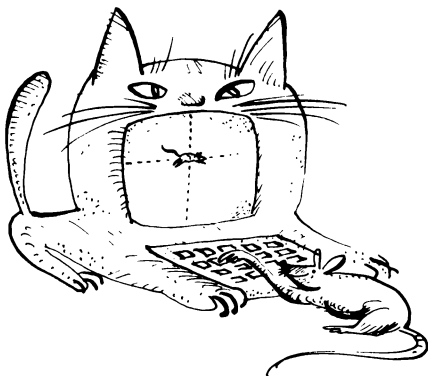
Frappez sur **ENTREE** pour étudier l'instruction LOCATE.

Mais vous pourriez d'abord faire avancer la petite flèche, en frappant sur **■** pour choisir une autre instruction.

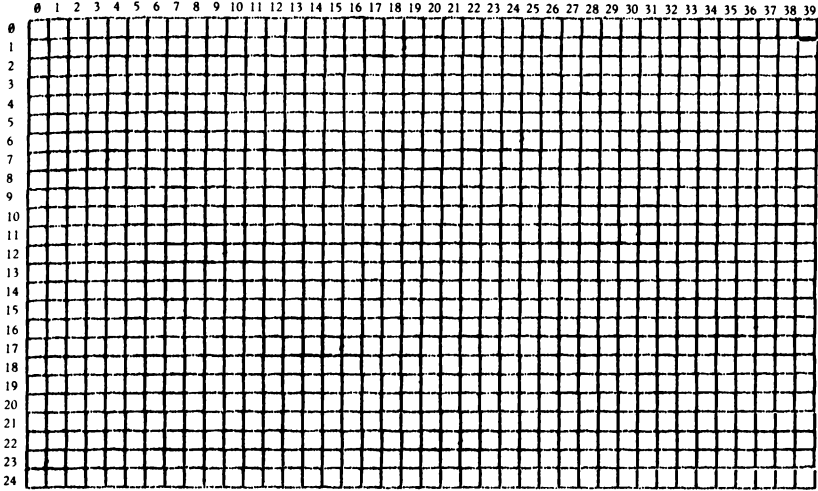
LOCATE - Définition

“LOCATE” signifie “placer”.

Sur l'écran du MO5 ou du TO7(-70), le curseur peut se placer à l'une des 1 000 positions possibles sur 25 lignes de 40 caractères.



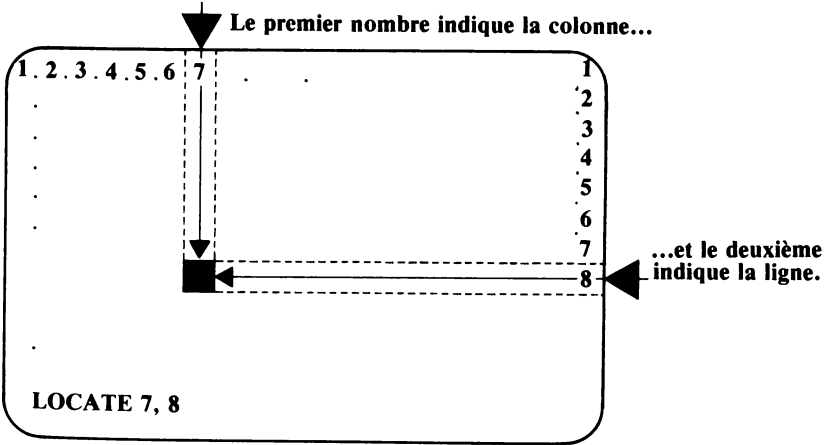
Voici une représentation de l'écran que vous pourrez reproduire et utiliser plus tard pour la "composition" de vos propres "pages".



Appuyez sur **ENTREE** pour lire la page suivante.

LOCATE - Syntaxe

Attention



Appuyez sur **ENTREE** pour lire la page suivante.

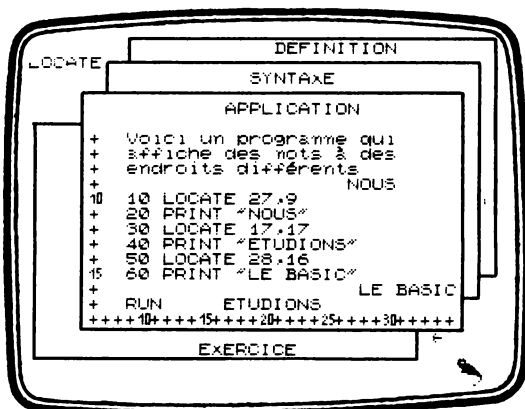


Remarque

Les pages “définition” et “syntaxe” sont des pages de lecture. Nous vous conseillons de les lire pour bien mémoriser la signification et la forme des instructions BASIC. Mais vous pouvez aussi “passer dessus” rapidement et travailler surtout les pages “application” et “exercice” qui sont plus animées.

Appuyez sur **ENTREE** pour passer à l’application.

LOCATE - Application



Les numéros de lignes et de colonnes indiqués sont les “vrais”. Ainsi, le rectangle vert “Application” va de la 6^e à la 35^e colonne.

Tout se passe comme si quelqu’un frappait un programme devant vous :

1 0 L O

Six instructions sont ainsi frappées.

Puis on a frappé **R U N ENTREE** , et le programme est exécuté par la machine ligne par ligne.

Le numéro de la ligne exécutée est colorié en rouge ; vous pouvez ainsi en voir immédiatement l’effet :

Par exemple : l’instruction 10 LOCATE 27,9 fait placer le curseur à la 27^e colonne de la 9^e ligne.

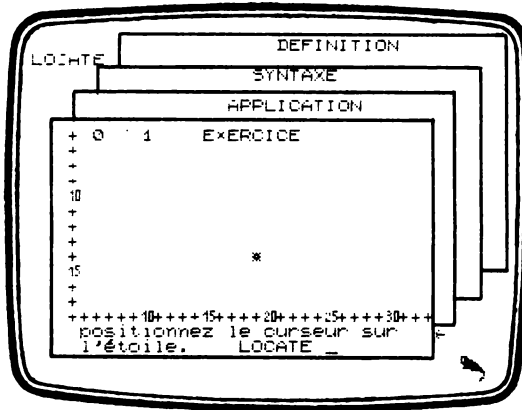
Attention

N'oubliez pas que vous pouvez appuyer sur **STOP** pour vous donner le temps de réfléchir à ce qui se passe.

Pour reprendre l'exécution il suffit d'appuyer sur la touche **■**.

Pour passer à l'exercice, appuyez sur **ENTREE** .

LOCATE - Exercice



Vous devez indiquer deux numéros séparés par une virgule et frapper **ENTREE** .

Exemple : **1 9 , 1 4 ENTREE**

Si une erreur vous est signalée, corrigez vous en utilisant la touche **■** .

Si vous souhaitez revoir la page SYNTAXE, par exemple, il vous faut frapper :

RAZ pour interrompre l'exercice.

■ pour placer la petite flèche devant la page qui vous intéresse.

ENTREE pour lire cette page.

Vous pouvez revenir à l'exercice en frappant **■** et **ENTREE** .

Si vous souhaitez passer à l'étude de l'instruction suivante, frappez **⤷** (comme si vous tourniez une page).

DÉFINITION

“PRINT” signifie “imprimer”. Lorsqu’une imprimante est branchée sur la machine, on peut, en effet, imprimer les caractères sur du papier grâce à cette instruction. Sur l’écran de télévision on parle plutôt “d’affichage”.

SYNTAXE

Voici intervenir les premiers “séparateurs” : la virgule et le point-virgule. Voici exactement ce qui se passe :

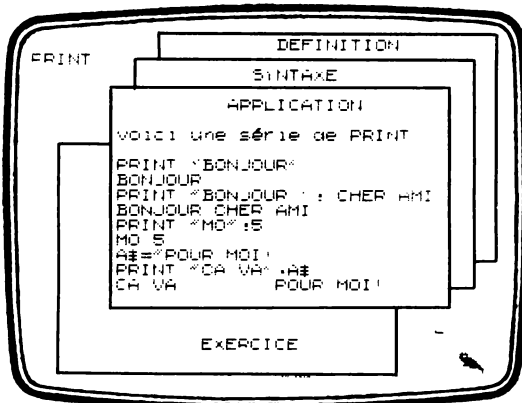
Lorsqu’une virgule précède un résultat à afficher, le curseur se place sur l’une des colonnes 0, 13, 26, ou 39, suivant immédiatement la position actuelle de ce curseur.

Cela permet d’avoir des affichages bien disposés verticalement sur 2 ou 3 colonnes lorsqu’on “édite” des suites de couples ou de triplets.

Par contre, la présence d’un point-virgule ordonne au curseur de ne pas avancer après l’affichage.

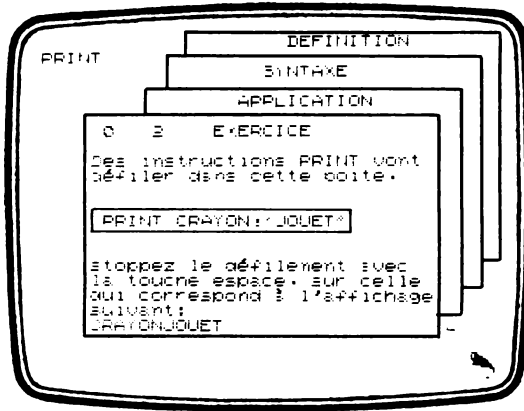
Enfin, l’absence de virgule ou de point-virgule à la fin de l’instruction PRINT entraîne le passage automatique à la ligne suivante.

APPLICATION



Observez attentivement l'effet des **■** et des **■**.

EXERCICE



En début d'exercice vous disposez d'une quinzaine de secondes pour réfléchir à chaque proposition de la machine ; mais le rythme s'accélère au fur et à mesure du déroulement.

Ne trichez pas en appuyant sur la touche **STOP** !

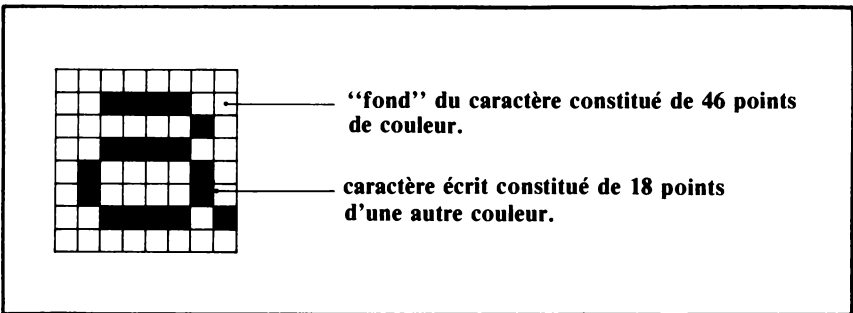
Pour interrompre l'exercice, n'oubliez pas l'une des touches **RAZ** ou **■**

DÉFINITION

“COLOR” signifie “couleur”.

Il faut bien comprendre la chose suivante :

Sur l'écran du MO5 ou du TO7(-70), chaque caractère est écrit sur un petit carré de 64 points (8 sur 8). Il est possible de choisir d'une part la couleur de ce petit carré constituant le “fond” de l'écriture et d'autre part, la couleur de l'écriture.



SYNTAXE

Le code des 16 couleurs du MO5 ou du TO7(-70) n'est pas trop difficile à retenir. 4 codes sont importants :

Ø noir, 1 rouge, 2 vert, 4 bleu.

(Pensez à la prise, dite RVB, de votre téléviseur et aux puissances de 2.)

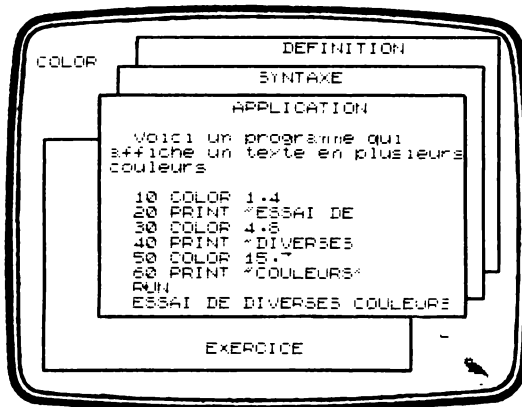
Ensuite (de 3 à 7) les couleurs sont obtenues par addition.

Ainsi : Violet = Rouge + Bleu

1 + 4 = 5

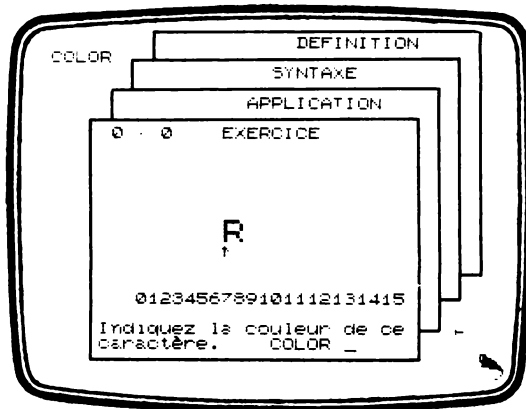
L'addition de 8 au code correspond à une couleur pastel plus claire ; sauf pour “blanc clair” ce qui ne serait pas très intelligent. 15 est donc le code d'une couleur supplémentaire : l'orange.

APPLICATION



Remarquez l'exécution pas à pas des instructions du programme et leur effet sur l'écran.

EXERCICE



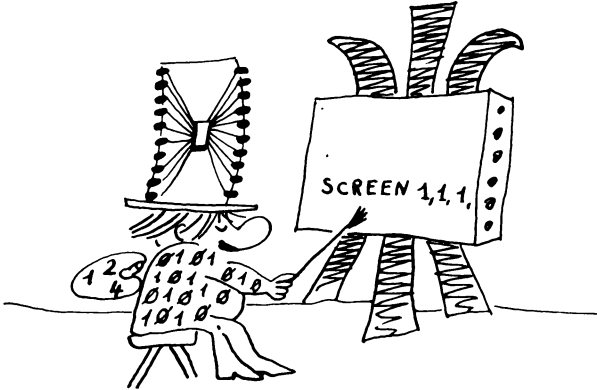
Vous devez indiquer les codes de deux couleurs : la couleur d'écriture et la couleur de fond de caractère (séparez ces codes par une virgule et n'oubliez pas de frapper **ENTREE** après ces deux codes).

Attention aux couleurs claires qui pimentent un peu la difficulté de l'exercice !

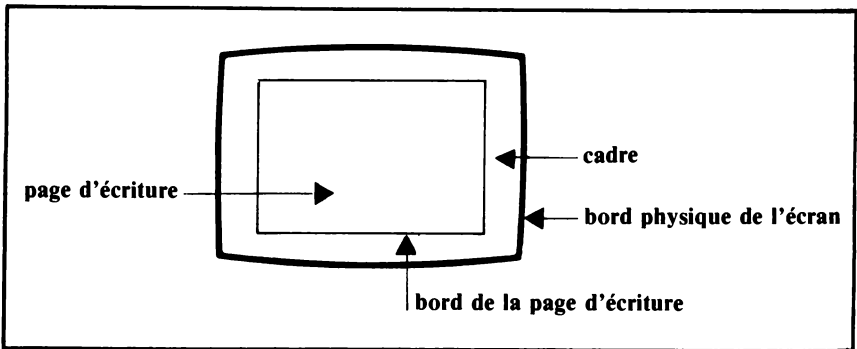
Pour vous aider à reconnaître les couleurs, leur code est rappelé sur l'écran, le fond de chacun étant justement le code correspondant.

DÉFINITION

“SCREEN” signifie “écran”.



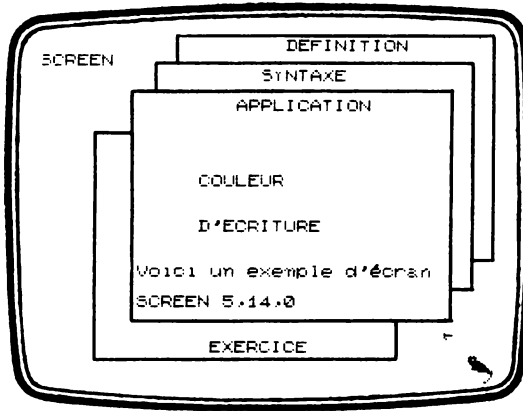
L'écran est divisé en 2 zones : le cadre dans lequel on ne peut pas écrire, et la page d'écriture.



SYNTAXE

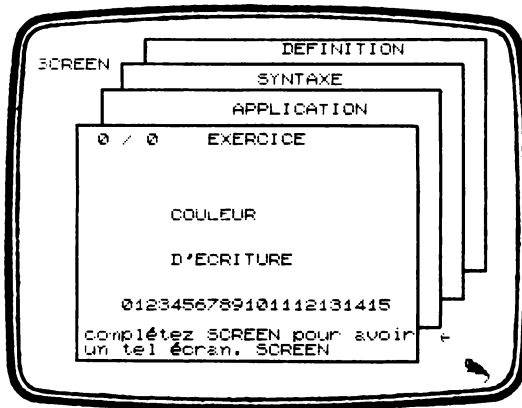
Normalement, le mot SCREEN est suivi de trois codes séparés par deux virgules. Vous verrez dans les compléments que bien d'autres syntaxes sont possibles.

APPLICATION



Un écran, dont les codes sont choisis au hasard par la machine, est donné en exemple.

EXERCICE



C'est à vous de trouver et d'écrire les codes des 3 couleurs « Screensques ». N'oubliez pas les virgules !



Le curseur invisible

Chaque fois que vous allez écrire quelque part sur l'écran, vous verrez le curseur se placer sous la position d'écriture prévue. Si vous ne souhaitez pas voir le curseur vous pouvez commander :

LOCATE 10, 20, 0

Les deux caractères **0** à la fin de l'instruction ordonnent au curseur de se cacher jusqu'à nouvel ordre. Il réapparaîtra lorsque vous terminerez une autre instruction LOCATE par **1**.

Des calculs dans LOCATE

Dans l'instruction LOCATE, les numéros de ligne et de colonne peuvent être le résultat de calculs.

Voici, par exemple, un programme qui peut vous donner l'idée de tracer des courbes si les mathématiques ne vous font pas trop peur.

```
10 LOCATE 2*K,K : PRINT "*" : K = K + 1 : GOTO 10
```

```
RAZ R U N 1 0 0 0 0 ENTREE
```



Attention

Évidemment, les petits programmes que nous vous proposons en ce début utilisent des instructions que vous n'étudierez que plus tard. Pour l'instant, n'y prêtez pas attention et considérez-les comme des mots à recopier... mais nous ne doutons pas que vous parviendrez à en tirer d'utiles enseignements.

Pour interrompre le déroulement d'un programme en cours d'exécution (qu'il ait été chargé à partir d'une cassette ou frappé par vous-même), vous pouvez par exemple : appuyer sur le petit bouton blanc en haut du clavier marqué "INITIAL. PROG."

Sur le MO5, ce (petit) bouton est rectangulaire et en haut à gauche du clavier... Sur le TO7(-70), il est carré et en haut à droite du clavier.

Noter que le programme interrompu est toujours en mémoire ; vous pouvez donc redemander son exécution.

Les limites de l'écran

Dans l'instruction LOCATE X,Y le nombre X doit être un entier compris entre 0 et 24 et Y entre 0 et 39.

Si X ou Y est négatif, une erreur est signalée.

Par contre, si X est plus grand que 24 (ou Y plus grand que 39) aucune erreur n'est détectée et le curseur se place tout en bas de l'écran (ou tout à fait à droite).

D'autre part, si X ou Y ne sont pas des nombres entiers la machine ne tient tout simplement pas compte de leur partie décimale.

Pour expérimenter ces divers cas, vous pouvez reprendre le programme précédent en remplaçant l'instruction LOCATE par l'une des instructions suivantes :

LOCATE 10 - K, K

LOCATE K/2, K

LOCATE K/3, K

LOCATE K, K * K/100

Le défilement de l'écran

Lorsqu'on a écrit sur la dernière ligne de l'écran, le passage à la "ligne suivante" provoque la montée de tout l'écran d'une ligne, la première étant alors "perdue".

Tout se passe comme si on écrivait sur un "rouleau" de papier qui défilerait devant une fenêtre.

Voici deux petits programmes expérimentaux montrant cet effet :

```
10 PRINT K : K = K + 1 : GOTO 10
```

```
20 LOCATE K, K : PRINT K : K = K + 1 : GOTO 20
```

(N'oubliez pas d'appuyer sur "INITIAL. PROG." pour interrompre l'exécution de ces programmes !)

Mettre ou ne pas mettre... des guillemets

La commande PRINT "A" fait afficher la lettre A.

La commande PRINT A fait afficher le nombre 0.

En effet, la lettre A est interprétée comme un nom de variable et, si vous n'avez donné aucune valeur à A, cette valeur est nulle et c'est ce qu'affiche donc la machine.

Essayez par exemple les "gags" suivants :

```
PRINT "ZERO", ZERO
```

```
PRINT "UN", UN
```

```
PRINT "7 × 8 =" ; 7 × 8
```

```
PRINT "2 + 2 =" ; 2 + 3 ; "OH PARDON !"
```

Couleurs de fond et d'écriture

L'instruction COLOR n'a pas d'effet sur les textes ou dessins déjà affichés sur l'écran. Elle fixe les couleurs qui seront utilisées lors du prochain affichage.

Voici quelques expériences intéressantes :

COLOR 1, 3 **ENTREE**

PRINT **ENTREE** (les caractères sont rouges sur fond jaune)

COLOR 3, 1 **ENTREE**

PRINT **ENTREE** (les caractères sont jaunes sur fond rouge)

COLOR 2 **ENTREE**

PRINT **ENTREE** (les caractères sont verts, le fond est toujours rouge)

COLOR , 4 **ENTREE**

PRINT **ENTREE** (les caractères sont toujours verts, le fond est bleu)

Notez que, pour se rendre compte de l'effet de COLOR, il faut écrire quelque chose. Ici on donne l'ordre PRINT qui fait aller le curseur à la ligne.

Quelques astuces sur la page d'écriture

L'instruction SCREEN a une double action : d'une part, elle réécrit l'ensemble des textes et dessins affichés sur l'écran en une seule couleur d'écriture sur un fond de couleur uniforme, d'autre part, comme COLOR, elle fixe les deux couleurs qui seront utilisées lors du prochain affichage.

EXPÉRIENCES

SCREEN 1, 2, 0 **ENTREE**

La page d'écriture est bien délimitée en vert entourée d'un cadre noir...

PRINT **ENTREE**

... et les caractères sont rouges.

SCREEN 2, 1 **ENTREE**

Le rouge et le vert se sont échangés d'un seul coup.

Il est alors assez drôle d'appuyer successivement sur **ENTREE** et **ENTREE** plusieurs fois de suite !

COLOR 1, 0 **ENTREE**

RAZ

COLOR 2, 1 **ENTREE**

La première ligne en haut de l'écran semble hors de la zone d'écriture, puisqu'elle est écrite sur un même fond que le cadre.

PRINT **ENTREE**

SCREEN 1, 2, 2, **ENTREE**

La page d'écriture semble élargie à l'écran entier lorsque le cadre et le fond sont de même couleur.

SCREEN 1, Ø, Ø **ENTREE**

Si vous devez travailler longtemps, le fond noir est certainement moins fatigant pour les yeux !

L'encre sympathique

Parmi les multiples possibilités de choix des couleurs d'écriture et de fond il en est une, apparemment stupide, qui réserve de bonnes surprises : celle qui consiste à écrire avec un "pinceau" de la même couleur que le "papier".

Évidemment, on n'y voit rien au moment de l'affichage. Mais, pour en être invisibles, les choses n'en sont pas moins écrites ; et elles redeviendront visibles dès que l'on commandera un changement instantané des couleurs par SCREEN.

EXPÉRIENCE

COLOR 4, 4 **ENTREE**

ME REVOILA ! **ENTREE**

Pendant la frappe des 11 caractères précédents vous ne voyez absolument rien et vous avez l'impression de frapper dans le vide. N'ayez pas peur de frapper l'instruction de "démasquage" suivante que vous ne verrez écrite qu'après la frappe de **ENTREE** .

SCREEN 4,6,6 **ENTREE**

Attention à ne pas faire d'erreur de frappe dans l'écriture de la commande SCREEN, car elle ne serait pas comprise par la machine et l'encre resterait "sympathique".



Remarque

Vous constatez, au démasquage, que la machine avait réagi par un message d'erreur, lorsque vous aviez frappé ME REVOILA ! En effet, ces mots ne correspondent à aucun ordre interprétable pour elle.

Les paramètres sous-entendus

Lorsque vous n'indiquez pas de valeurs pour les codes de couleurs dans une instruction SCREEN ou COLOR, la valeur sous-entendue reste celle qui est alors "en cours". On dit que l'on garde la "valeur courante".

EXPÉRIENCES

SCREEN 4, 6, 6 **ENTREE**

SCREEN , , 0 **ENTREE**

Seul le cadre a changé de couleur.

SCREEN , 0, 6 **ENTREE**

L'écriture a gardé sa couleur.

SCREEN 1, , 7 **ENTREE**

La page d'écriture a gardé sa couleur.

COLOR , 2 **ENTREE**

PRINT **ENTREE**

L'écriture est restée rouge mais le fond de caractère est maintenant vert.

COLOR 0 **ENTREE**

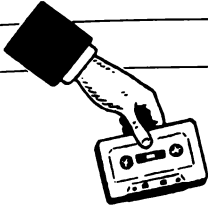
SCREEN 4, 6 **ENTREE**

Lorsque le nombre de paramètres est inférieur à celui que la machine attend, les derniers (non écrits) gardent leur valeur courante.

CHAPITRE 2

GRAPHIQUE ET SON

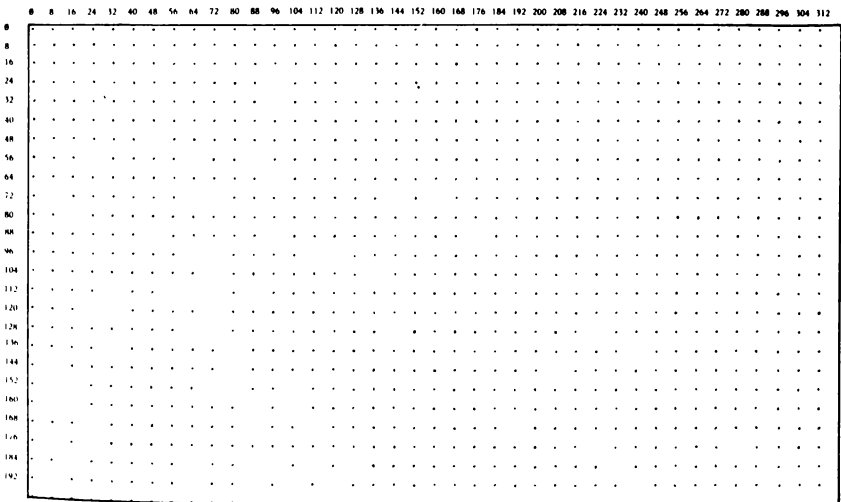
Suivi de la cassette



Dans le chapitre 1, l'écriture d'un caractère sur l'écran pouvait se réaliser sur chacune des 1 000 positions possibles ; plus précisément sur l'une des 40 colonnes de l'une des 25 lignes.

Un caractère étant constitué de petits points, (exactement de 8 fois 8 points), il est en fait possible de marquer des points en 64 000 positions différentes sur l'écran ; plus précisément en l'une des 320 abscisses-colonnes possibles et l'une des 200 ordonnées-lignes possibles.

Les 64 000 points de l'écran en "mode-graphique" :

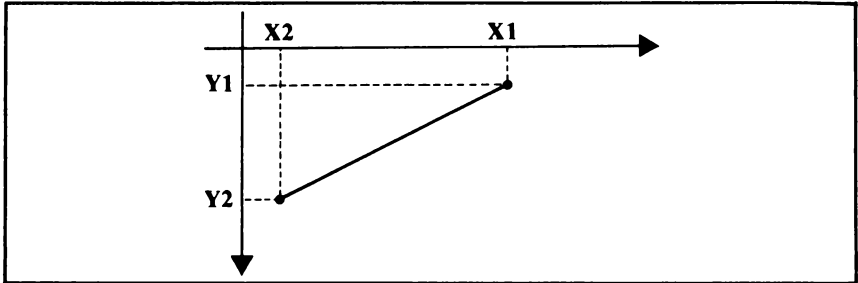


Un caractère :



“LINE” (anglais) signifie “ligne”, mais il vaudrait mieux traduire par “segment”.

Un segment est défini par ses deux extrémités et chaque extrémité par ses deux coordonnées. Pour tracer un segment sur l'écran on doit donc donner 4 nombres (plus 1 nombre pour la couleur du tracé).

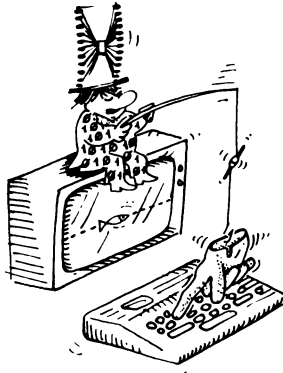


Attention

Les axes de coordonnées liés à l'écran du MO 5 ne sont pas les axes habituels des mathématiques du collège : l'axe vertical est orienté vers le bas. Sa direction est en effet liée au “sens de la lecture” c'est-à-dire du haut vers le bas.

De même, puisqu'on lit d'abord de gauche à droite puis ensuite de haut en bas, on indiquera d'abord le numéro de colonne, puis le numéro de ligne.

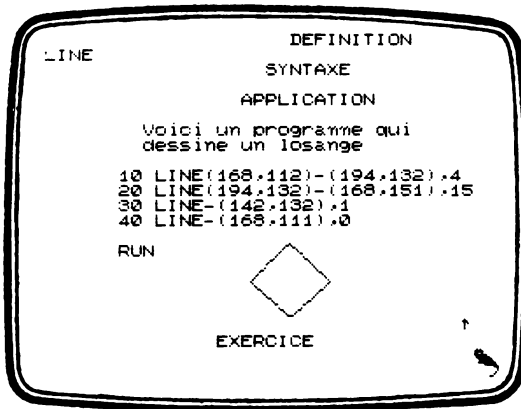
Attention, il y a 2 pages syntaxe pour cette instruction !



On peut en effet tracer un segment en mode-graphique, sur 320×200 points ou bien en mode-caractère sur 40×25 positions.

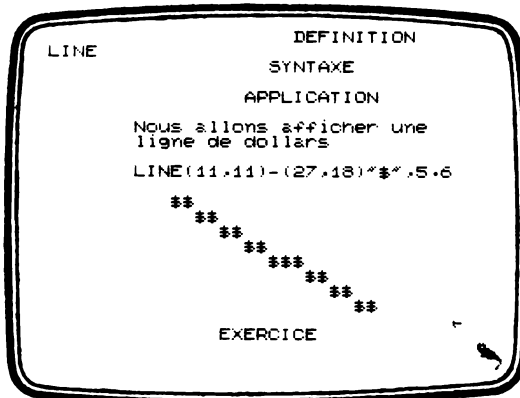
Le MO5 ou le TO7(-70) reconnaît qu'il n'est pas en mode-graphique si un guillemet (") suit l'indication des coordonnées ; il est alors en mode-caractère c'est-à-dire que d'une part, il interprétera les numéros de ligne et de colonne comme des positions de caractères, d'autre part il marquera (sur la position indiquée) le caractère indiqué après le guillemet.

Attention, il y a aussi deux pages application pour cette instruction !



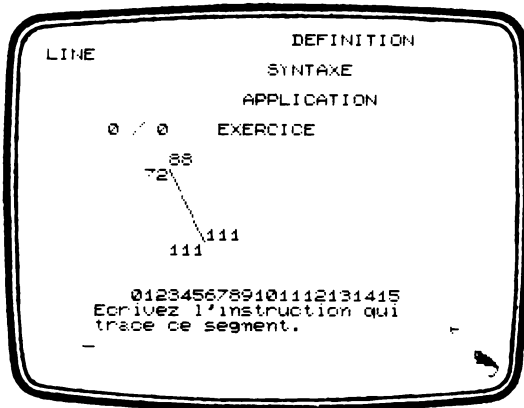
L'exemple donné en mode-caractère vous montre le genre de défauts que peut présenter le tracé d'une ligne lorsque les "points" utilisés pour le tracé sont trop "gros" pour la qualité de notre vue.

Observez bien l'effet des numéros codes de couleur.



EXERCICE

Les exercices vous seront proposés au hasard en mode-caractère ou en mode-graphique.



Vous devez écrire l'instruction complète, avant de frapper sur **ENTREE**. Par exemple :

LINE (88,72) – (111,111), 12 **ENTREE**

N'oubliez pas la couleur !

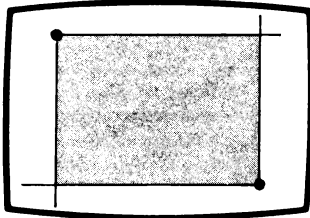
Les nombres marqués près des extrémités du segment vous aident dans la détermination des coordonnées.

Attention à l'ordre !

BOX BOXF

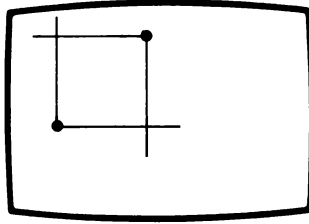
“BOX” signifie “boîte”.

“BOXF” est l'abréviation de “BOX Full” qui signifie “boîte pleine”.

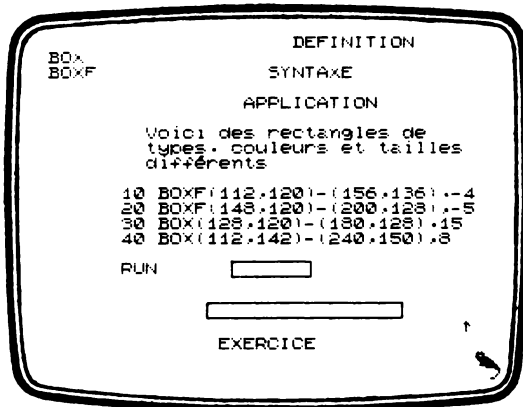


Peut-être ne l'aviez-vous jamais remarqué, mais étant donné 2 points,

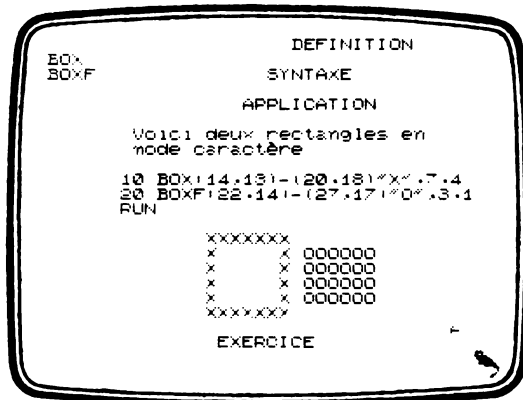
il n'existe qu'un rectangle ayant ces 2 points pour sommets et dont les côtés sont parallèles au bord d'une feuille de papier (ou de l'écran).



La syntaxe est la même que pour l'instruction LINE. (Il y a aussi 2 pages-syntaxe, car il est possible de tracer les côtés d'un rectangle ou de le remplir avec des caractères).



L'effet de la couleur de fond ne se voit pas (-4 semble équivalent à 3) mais cela aura une certaine importance (voir compléments).



“PSET” est l’abréviation de “point” suivi de “SET” qui signifie “placer”. Il s’agit donc de “placer un point”.

Si vous ne l’avez pas déjà fait, c’est le moment de regarder votre écran à la loupe ; vous y verrez comment les trois couleurs (rouge, vert et bleu) s’y groupent pour dessiner des points de couleur.

Attention, il y a encore 2 pages syntaxe.

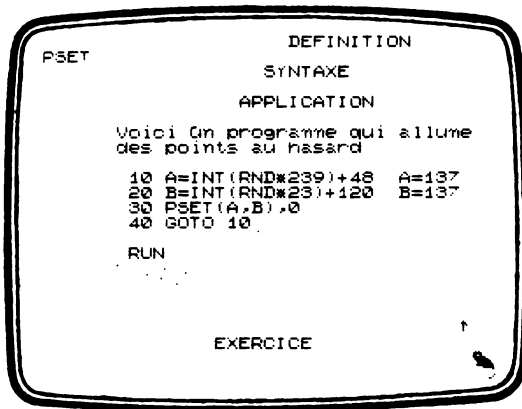
L’instruction :

PSET (C,L) “n”, A, B,

semble avoir le même effet que les 3 instructions :

COLOR A, B : LOCATE C, L : PRINT “n”

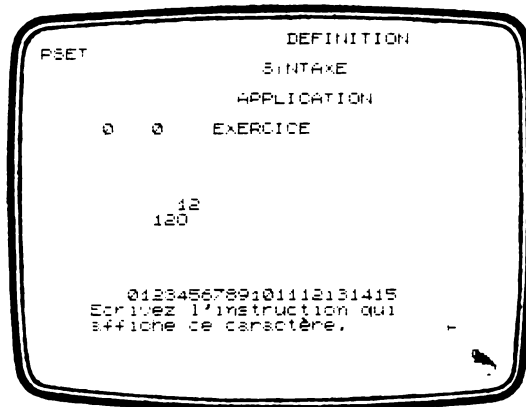
Mais leur avenir n’est pas tout à fait le même... (voir les compléments).



Ne faites pas trop attention aux instructions 10 et 20 qui tirent des coordonnées de points au hasard (vous saurez tout sur ce tirage dans le chapitre 12).

EXERCICE

Pour vous permettre de mieux repérer le “point” affiché, l’exercice utilise le mode-caractère et les numéros de ligne et colonne sont affichés comme aide supplémentaire.



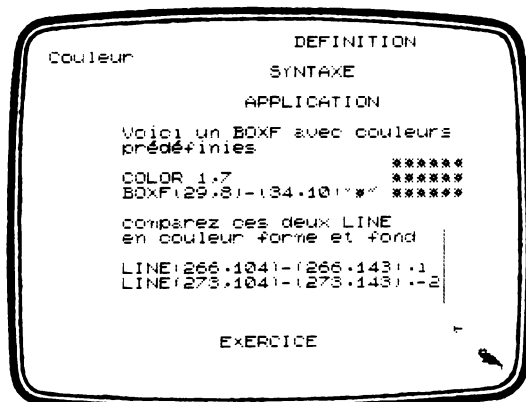
Pour répondre, il faut frapper l'instruction complète, par exemple : PSET (12,12), "o", 4,5 **ENTREE**

Couleur

Comme pour la plupart des instructions déjà rencontrées (voir COLOR et SCREEN au chapitre 1), les paramètres de couleur en fin d'instruction peuvent ne pas être indiqués. Dans ce cas, ce ou ces paramètres gardent leur valeur "en cours", dite "valeur courante".

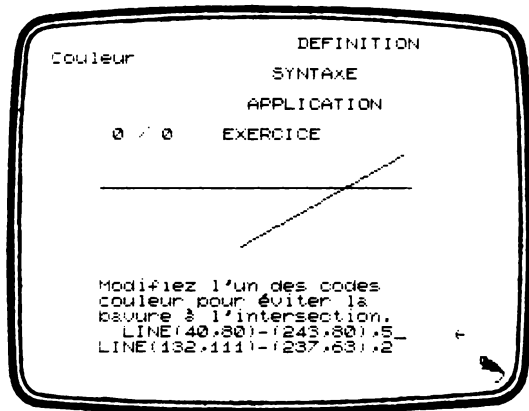
Lorsqu'on parle de la couleur de fond – F, il s'agit de la couleur de numéro F – 1.

Lorsqu'on veut colorier un fond avec la couleur de numéro C, il faut parler de la couleur de fond – (C + 1).



Pour bien comprendre l'effet des deux instructions LINE, il faut remarquer que 272 est un multiple de 8 et lire le paragraphe "les bavures de couleurs" dans les compléments.

EXERCICE



Il vous est demandé de ne modifier que l'un des deux codes.

Si vous ne voulez pas modifier la première ligne frappez **ENTREE** (n'oubliez pas la touche **■** pour déplacer le curseur).

PLAY

"PLAY" signifie "jouer".

Vous savez certainement qu'un son pur est caractérisé par 4 paramètres au moins :

— La *fréquence* des vibrations de l'air qui le transmet.

Toutes les fréquences ne sont pas émissibles ; seuls les sons qui portent un nom de note dans la gamme tempérée (DO, DO# ou RÉb, RÉ..., LA# ou SIb, SI) sont "jouables".

Ce sont donc l'octave et le nom de la note qui fixent la fréquence d'émission.

— *L'intensité* ; c'est le bouton de réglage du volume de votre téléviseur qui fixe ce paramètre.

— La manière dont l'intensité est modulée pendant la durée d'émission de la note ; c'est le paramètre *d'attaque* qui caractérise cette manière.

— La *durée* pendant laquelle la note est émise.

En fait, dans la notation classique de la musique, la durée d'émission d'une note unique n'a pas grand sens.

Ce qui importe c'est le rythme créé par l'émission d'une suite de notes. D'où l'apparition de deux paramètres :

— Le rythme général de la suite de notes, défini, par exemple, en fixant la durée d'une note particulière (dite "noire") ; c'est le *tempo*.

— La durée occupée individuellement par chacune des notes de la suite, généralement obtenue par division ou multiplication par 2 ou 3 à partir de la noire ; sur le MO5 ou le TO7(-70), c'est le paramètre qui s'appelle la *durée*.

Parmi les paramètres définissant un son, le MO5 ou le TO7(-70) permet donc de choisir :

. La fréquence par le choix du nom de la note et de l'octave.

Exemple : "O2 DO #"

. L'attaque.

Exemple : "A1 DO A100 RE"

. Le tempo.

Exemple : "T1 DO RE MI T50 DO RE MI"

. La durée.

Exemple : "L48 DO L96 RE L24 MI"



```

DEFINITION
PLAY
SYNTAXE
APPLICATION
NOUS ALLONS JOUER AU CLAIR
DE LA LUNE

10 PLAY "A0T4L2404D0D0D0REL4
3MIREL24DOMIREREL48DOP"
20 PLAY "L24D0D0D0REL48MIREL
24DOMIREREL48DOP"
30 PLAY "L24RERERERE03L48LAL
A04L24RED003SILAL48SOP"
40 PLAY "L2404D0D0D0REL48MIR
EL24DOMIREREL48D0PL24"
RUN

EXERCICE

```

Observez bien les changements de durée à l'intérieur de la "chaîne" des notes.

L24 : noire.

L48 : blanche.

EXERCICE

```

DEFINITION
PLAY
SYNTAXE
APPLICATION
EXERCICE

Voici une mélodie:
PLAY "DOREMIREMIFAFASOMIREDO"

Modifiez à votre gré les
paramètres suivants.

TEMPO (1-255) 5 taper
OCTAVE (1-5) 4 E pour
ATTQUE (0-255) 0 écouter
DUREE (1-96) 24

```

Vous allez pouvoir vous livrer à toutes les expériences possibles sur le MO5 ou le TO7(-70). Pour bien apprécier l'effet de chaque paramètre, il vaut mieux n'en modifier qu'un à la fois. (La frappe sur **ENTREE** permet de passer d'un paramètre au suivant.)

Attention

Si vous choisissez un tempo égal à 255, vous devez écouter jusqu'au bout l'émission des 11 notes à un rythme très très très très lent. Et il n'y a aucun moyen d'interrompre l'exécution (hormis le bouton "INITIAL. PROG." qui vous oblige à relancer le logiciel par RUN). Essayez donc d'abord des tempos de 10, 20,...



Des cadres assortis

La “planche à dessin” du MO5 ou du TO7(-70) mesure 320 sur 200 mm.

Remarquez d’ailleurs que ces mesures assurent qu’une boîte de même mesure verticale et horizontale est un carré. C’est la moindre des choses, direz-vous, mais ce n’est pourtant pas le cas avec tous les micro-ordinateurs.

EXPÉRIENCE

BOXF (0,0) – (100,100)

Il est souvent agréable de délimiter le cadre de travail par une fine ligne, ou bien de détacher une ligne de “titre de page” tout en haut. Essayez par exemple l’une des deux présentations ci-dessous :

EXPÉRIENCE

10 SCREEN 1,0,0

20 BOX (0,0) – (319,199)

30 LOCATE 17,0 : ? “TITRE = PAGE1”

100 SCREEN 1,0,0

110 LINE (0,7) – (319,7) : LINE (0,99) – (319,199)

120 LOCATE 0,0 : ? “TITRE PAGE1”

Les rectangles de dimensions semblables à la planche à dessin complète sont certainement plus agréables à l’œil que d’autres.

Sachez donc que 320×200 est “semblable” à :

64×40

16×10

8×5

ACTIVITÉ

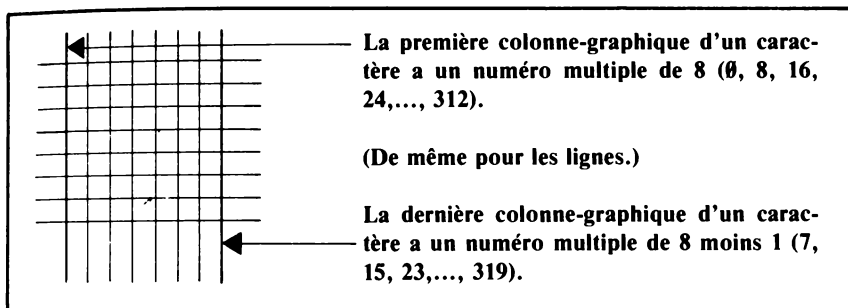
Essayez de dessiner de tels rectangles dans les coins ou centrés au milieu de l’écran !

Respectez les multiples de 8 !

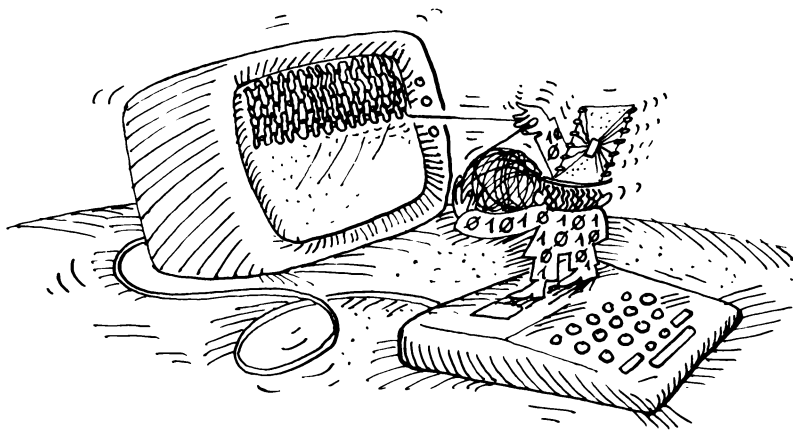
Un avantage du MO5 ou du TO7(-70) est de pouvoir mêler sur l'écran des textes et des dessins.

Cependant les textes sont repérés dans un système $(0,39) \times (0,199)$. Il est donc important de s'habituer au respect et à la considération des multiples de 8 puisqu'un caractère est bâti sur un carré 8×8 .

Le schéma suivant est assez utile :



Par exemple, l'encadrement le plus "serré" d'un texte par un filet de couleur utilise obligatoirement les multiples de 8.



EXPÉRIENCE

10 COLOR 1 : LOCATE 15,10 : ? "BRAVO"

20 BOX (15*8 - 1,10*8 - 1) - (160,88)

15*8 + 40 vaut 160.

Plus généralement, nous vous conseillons de préparer vos dessins sur une grille analogue à celle que nous vous avons offerte page 37... et de vous habituer à bien placer les points particuliers de vos dessins par rapport aux multiples de 8. Faites nous confiance : vous y gagnerez du temps et de l'esthétique.

Le carton de fond, lui, inverse l'utilisation des 8K et des 8K - 1.

EXPÉRIENCE

10 BOXF (120,80) - (159,87), - 1

20 COLOR 1 : LOCATE 15,10 : ? "BRAVO"



Attention

Si vous dessinez la boîte après avoir écrit, le texte est effacé par le dessin de la boîte.

Les couleurs en cours

L'instruction COLOR 1 fixe la couleur d'écriture à « rouge » pour toute écriture future avant un autre changement de couleur.

L'instruction LINE (0,0) - (80,50), 1 dessine un segment en rouge mais n'a aucun effet sur la couleur d'écriture future.

EXPÉRIENCE

Analyser, par exemple, l'effet des instructions suivantes.

10 COLOR 1 : BOX (0,0) - (128,80) : ? "A"

De quelle couleur est la boîte ?

20 BOX (8,8) - (112,64) , 2 : ? "B"

De quelle couleur est le B ?

30 BOX (16,16) - (96,48)

De quelle couleur est la boîte ?

40 COLOR 3

50 BOX (24,24) - (80,32) : ? "D"



Attention

En mode caractère les instructions BOX, BOXF, LINE et PSET changent les couleurs courantes.

Effets de "cartes"

Le coloriage de plages décalées sur l'écran permet de jolis effets. La cassette d'auto-initiation utilise beaucoup cette technique. On peut chercher soit la régularité du décalage, soit une certaine diversité dans le "jet" des "cartes".

EXPÉRIENCES

- 1Ø BOXF (Ø,Ø) – (127,79),1
- 2Ø BOXF (16,16) – (143,95),2
- 3Ø BOXF (32,32) – (159,111),3

Comparez, par exemple, les effets produits par les programmes ci-dessus et ci-dessous.

- 1ØØ BOXF (Ø,Ø) – (127,79),1
- 11Ø BOXF (8Ø,48) – (199,169),2
- 12Ø BOXF (32,64) – (143,183),3

Des boîtes en couleur de fond

Vous verrez au paragraphe suivant que les notions de "fond" et de "forme écrite sur ce fond" ne sont pas si simples qu'il y paraît. (Mais Platon ne le signalait-il pas déjà à ses contemporains ?)

Tant que vous n'écrivez rien dans une boîte vous n'avez pas à vous soucier de la manière dont vous la coloriez. Mais si vous la coloriez en couleur d'écriture, puis que vous écriviez dedans, alors la couleur de la boîte se transforme (caractère par caractère) en couleur de fond. Il se peut alors que certains bords de caractère laissent apparaître la nouvelle couleur d'écriture.

EXPÉRIENCE

Comparez les effets de :

1Ø BOXF (Ø,Ø) – (8Ø,5Ø),3 : LOCATE 1,1 : ? "Ø"
et

2Ø BOXF (1ØØ,1ØØ) – (18Ø,15Ø), – 4 : LOCATE 1,1 : ? "Ø"

Un conseil donc :

Si vous dessinez des boîtes dans le but d'écrire à l'intérieur, pensez à les colorier en couleur négative.



Remarque

Plutôt que de dessiner des boîtes graphiques pleines de points, vous serez un jour intéressé par un dessin qui paraît équivalent : des boîtes caractères pleines d'espaces.

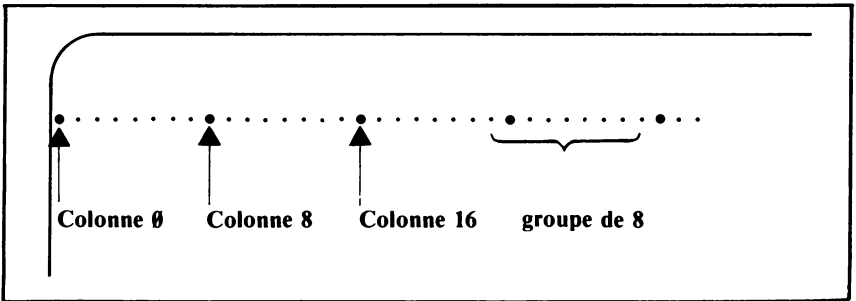
EXPÉRIENCE

10 BOXF (10,10) – (18,15) “ ”, – 4

D'ailleurs, si vous voulez écrire dans une boîte, une telle instruction vous évite de faire attention aux multiples de 8.

Les “bavures” de couleurs

Pour des raisons d'économie de mémoire, chacun des 64 000 points de l'écran ne peut pas être colorié d'une couleur absolument quelconque. Précisément, chaque groupe de 8 points sur une ligne horizontale constitue un ensemble pour lequel on ne peut utiliser que 2 couleurs (et non 8 couleurs).



Pour chaque groupe de 8 points horizontaux d'une même position de caractères :

- Deux couleurs peuvent être choisies ; l'une est la couleur de fond, l'autre la couleur d'écriture.
- Pour chaque point du groupe, un codage en mémoire indique s'il s'agit d'un point de fond ou d'écriture.
- Lorsque l'un des points d'un groupe change de couleur, ce sont

tous les points du même groupe et de même nature (fond ou écriture) qui prennent la nouvelle couleur.

Les quelques instructions suivantes valent la peine d'être exécutées car, finalement, cette histoire de fond et d'écriture n'est pas si simple que cela !

EXPÉRIENCES

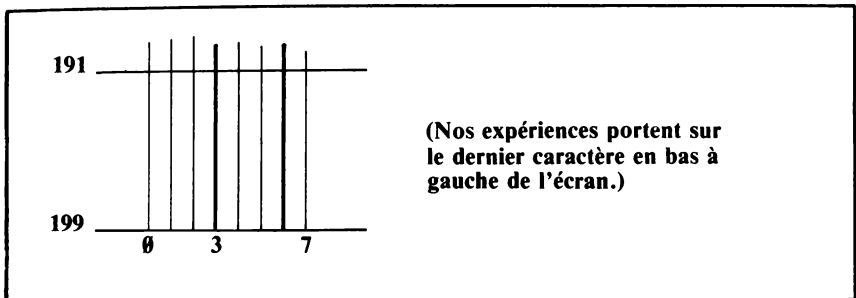
RAZ

SCREEN 1, Ø, Ø

LINE (191,3) – (199,3)

Une verticale rouge sur un fond noir

LINE (191,6) – (199,6),2



Une verticale verte sur un fond noir. Mais le changement de couleur a porté sur l'ensemble des colonnes du caractère de Ø à 7 : la verticale rouge est devenue verte.

LINE (191,Ø) – (199,Ø), – 5

Le changement de la première colonne en couleur de fond bleu a entraîné le changement de tous les autres points de fond des mêmes groupes de huit.

Il y a maintenant deux verticales vertes sur un fond bleu.

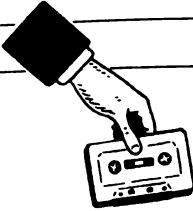
LINE (191,Ø) – (199,Ø)

La première verticale est devenue de l'écriture et les points d'écriture concernés se sont coloriés de la couleur courante (rouge).

En ce qui concerne la couleur, les points sont donc liés horizontalement 8 par 8 ; c'est une raison supplémentaire de surveiller les multiples de 8 lorsqu'on travaille sur les coordonnées de points de l'écran.

CHAPITRE 3

DONNEES ET VARIABLES



Suivi de la cassette

En informatique, les nombres ou les mots que l'on désire transformer ou conserver doivent être rangés dans des mémoires (que l'on peut imaginer comme des boîtes ou des tiroirs). Les noms qui permettent de retrouver ces nombres ou ces mots ne sont pas leurs noms à eux mais le nom du tiroir ("mémoire") où ils se trouvent à un moment donné.

Ainsi l'instruction :

$$A = B + C$$

commande la suite d'actions suivantes :

- Aller chercher le contenu du tiroir B.
- Aller chercher le contenu du tiroir C.
- Effectuer l'addition.
- Rangèr le résultat dans le tiroir A.

Si l'on comprend bien ces manipulations, on n'est plus étonné d'une instruction du type :

$$A = A + 1$$

Les actions déclenchées par l'exécution sont successivement :

- Aller chercher le contenu du tiroir A.
- Lui ajouter 1.
- Placer le résultat dans le tiroir A.

Ainsi l'exécution de l'instruction $A = A + 1$ fait-elle simplement augmenter de 1 le contenu de A.

Retenez donc que, dans notre contexte, le mot VARIABLE est employé dans le sens du mot TIROIR.

C'est pourquoi l'on dira souvent "la variable B contient la valeur 365" plutôt que "B prend la valeur 365".

Les données

Les données numériques sont des nombres, comme on en a l'habitude.

Les données alphanumériques sont des mots que l'on peut mettre bout à bout ("concaténer" vient de "catena" signifiant "chaîne") ou bien composer selon l'ordre alphabétique.

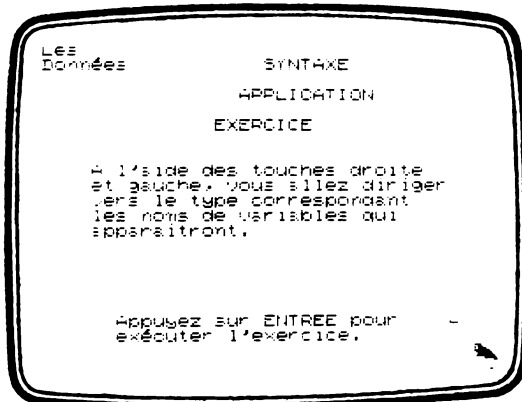
Mais on peut aussi extraire des morceaux d'un mot ou contrôler si une certaine suite de caractères est contenue ou non dans un mot.

Les mémoires dans lesquelles sont rangés les nombres et les mots se distinguent à la fin de leur nom. Un "dollar" (\$) caractérise les noms de mémoires contenant des mots.

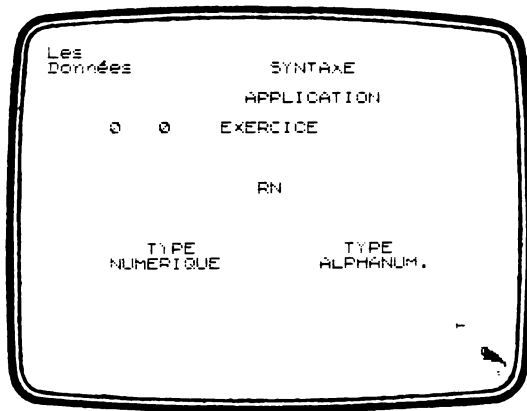
Attention

Les noms de variables ne peuvent pas commencer par un chiffre !

EXERCICE



Après l'apparition de la première page, appuyez sur **ENTRÉE** .



Le nom de la variable (ici RN) descend vers le bas de l'écran. RN étant de type numérique, il vous faut appuyer plusieurs fois sur la touche **■**.

Si, par exemple, le nom de variable Z\$ apparaît, il vous faudra appuyer sur **■**.

Données numériques

Les mémoires des ordinateurs d'aujourd'hui sont organisées en "octets" (c'est-à-dire 8 bits).

Il est donc simple de ranger les nombres entiers dont l'écriture en base 2 comporte 15 chiffres ou plus (1 bit étant utilisé pour le signe + ou -).

Or $2^{15} = 32\ 768$.

Lorsqu'on signale à la machine qu'un nombre est entier et plus petit que 32 768, elle peut donc le ranger dans deux mémoires (c'est-à-dire 16 bits).

Dans le cas contraire le nombre manipulé, qui peut être un nombre décimal, est appelé un nombre réel. Son rangement utilise beaucoup plus de mémoires ; mais de toutes façons, on ne peut pas dépasser le nombre 10^{38} (1 suivi de 38 zéros) qui est la valeur approximative de $2^{(2^7)}$ (c'est-à-dire 2^{32768}).

Attention

Le TO7(-70) peut conserver jusqu'à 15 chiffres significatifs à condition que le nom du nombre conservé se termine par un # . Le MO5 lui ne conserve pas plus de 6 chiffres significatifs (comme le TO7 lorsque le nom ne se termine pas par un dièse).

Ainsi le nombre 52 328 243 est conservé sous la forme 52 328 200 et il sera affiché sous la forme 5.23282 E7 (c'est-à-dire "5,23282 la virgule étant déplacée de 7 positions vers la droite").

A% est un " tiroir de rangement de nombres " comprenant 2 octets.

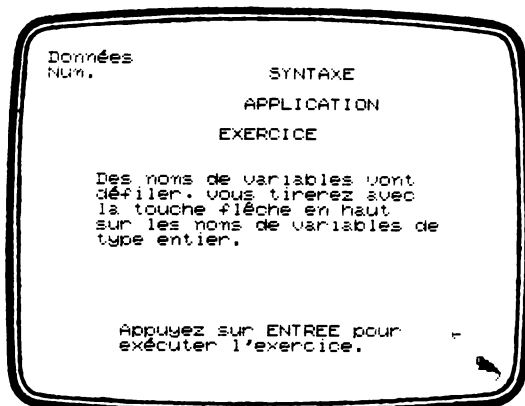
A est un " tiroir de rangement de nombres " comprenant 6 octets.

C'est donc pour des raisons évidentes d'économie de place que l'on donne des noms terminés par %

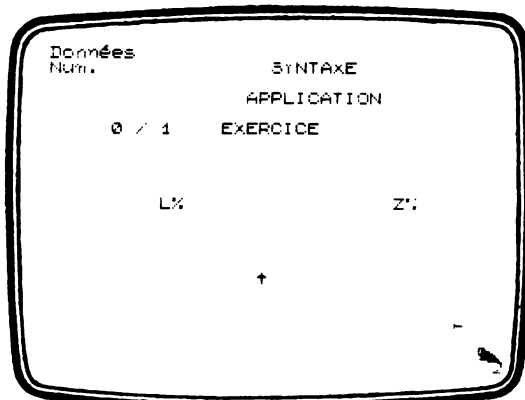
Le symbole de la division est /, et non par ÷.


De même le symbole de la multiplication est *.

EXERCICE



Après l'apparition de la première page appuyez sur **ENTREE** .



Les noms de variables se déplacent de la gauche vers la droite. Lorsqu'ils se terminent par un % (type entier), il vous faut les empêcher de passer en tirant dessus lorsqu'ils sont au-dessus de votre canon. Pour cela, appuyez sur . Ici l'utilisateur a laissé passer Z%, il a donc un score de 0 sur 1.

Données alphanumériques

Dans le contexte informatique, les “suites” de caractères sont plutôt appelées des “chaînes”.

Si on écrit $A = 365$, alors le tiroir A contient l'écriture binaire du nombre “trois cent soixante-cinq” (c'est-à-dire exactement 101100101).

A est une variable numérique.

Par contre, si l'on écrit $A\$ = “365”$, alors le tiroir A\$ contient la suite des codes de chaque chiffre 3, 6 et 5 (c'est-à-dire exactement 110011, 110110 et 110101).

A\$ est une variable alphanumérique.

Dans une instruction, on distingue un “nombre” d'une “chaîne” par les guillemets qui encadrent la suite de caractères composant la chaîne.



Attention

Il n'est pas possible d'écrire quelque chose comme :

$A\$ = \text{ANDRÉ}$

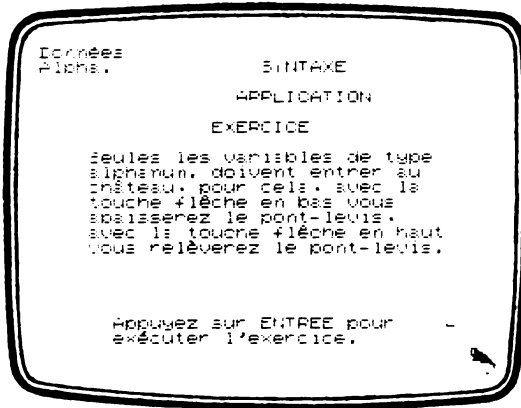
ANDRÉ est en effet interprété comme le nom d'une variable numérique (c'est-à-dire le nom d'un tiroir contenant un nombre, d'ailleurs certainement nul) et une erreur de syntaxe est détectée.

Les chiffres sont des caractères particuliers et vous pouvez donc les mettre entre guillemets ; mais alors ils ne feront l'objet d'aucun traitement opératoire.

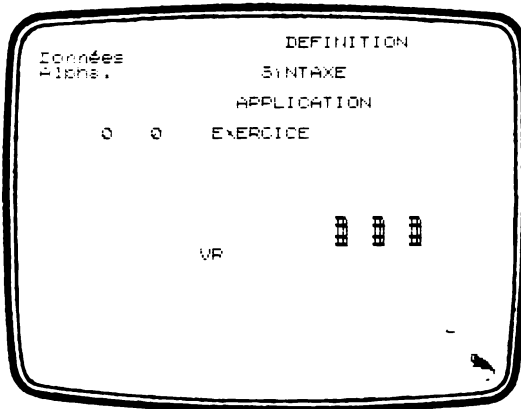
“14” est la suite de chiffres “1 suivi de 4”.

14 est le nombre quatorze.

EXERCICE



Après l'apparition de la première page, appuyez sur **ENTREE** .



La variable VR n'étant pas alphanumérique, elle ne doit pas rentrer dans le château ; il vous faut donc relever le pont-levis en appuyant sur **↑** .

Si la variable VR\$ se présentait, il faudrait au contraire le baisser en appuyant sur **↓** .



L'affectation

Le fonctionnement du signe “=” paraît toujours inhabituel à ceux qui font ordinairement de l’algèbre. Ainsi, le fait de faire exécuter l’instruction $A = B$ ne signifie pas du tout que dans la suite du programme A et B seront “égaux” ; cela signifie que, au moment de l’exécution de l’instruction et à ce moment là seulement, A prend la valeur qui est actuellement celle de B . Ce qui se passe ensuite est une autre histoire.

Afin de mieux illustrer ce fonctionnement, livrez vous aux quelques expériences suivantes qui montrent bien que les “instructions d’affectation” n’ont rien de “commutatif”.

EXPÉRIENCES

10 $A = 1 : B = 1$

20 $C = A + B : PRINTC : A = B : B = C : GOTO 20$

100 $A = 1 : B = 1$

110 $C = A + B : PRINTC : B = C : A = B : GOTO 110$

L’un des programmes affiche les puissances de deux, l’autre affiche une suite dite “de Fibonacci”.

L’adaptation de ces deux programmes avec des variables alphanumériques donne de drôles de résultats ; jugez-en :

200 $A\$ = “TA” : B\$ = “TI”$

210 $C\$ = A\$ + B\$: PRINTC\$: A\$ = B\$: B\$ = C\$: GOTO 210$

300 $A\$ = “TA” : B\$ = “TI”$

310 $C\$ = A\$ + B\$: PRINTC\$: B\$ = C\$: A\$ = B\$: GOTO 310$

Enfin, en remplaçant $A\$ = “TA” : B\$ = “TI”$ par $A\$ = “O” : B\$ = “+”$ ou bien par $A\$ = “-” : B\$ = “.”$, on voit mieux les différences entre les affichages - résultats.

EXPÉRIENCES

Si vous n’avez jamais affecté de valeur à une variable, cette variable contient la valeur \emptyset .

Ainsi :
10 PRINT COMBIEN
(La variable COMBIEN a la valeur 0)

Gag

100 ZERO = 1
110 PRINT ZERO

L'échange de valeurs

Supposons que A vaille 3 et que B vaille 5.
Vous voulez échanger les valeurs de A et de B.

 **Attention**

Il ne vous faut surtout pas écrire les instructions :
A = B : B = A

En effet, après la première affectation, A vaut 5 (et la valeur 3 est donc perdue).

Il vous faut utiliser une variable auxiliaire ("fantôme" qui ne fera que passer) pour stocker momentanément la valeur de A. Comme ceci :

C = A : A = B : B = C

... et le tour (c'est le cas de le dire) est joué !

Petits et grands nombres

Le BASIC du MO5 ou du TO7(-70) limite à 6 le nombre de chiffres significatifs conservé en mémoire. De sorte que "un million + un" n'est pas différent de "un million".

On peut alors obtenir de drôles de choses (à faire dresser les cheveux sur la tête d'un mathématicien).

EXPÉRIENCE

Composer le résultat de chacun des deux programmes :

10 X = 1000000 : Y = 1 : PRINT X + (Y - Y)

20 X = 1000000 : Y = 1 : PRINT (X + Y) - Y

ACTIVITÉS

Grâce à un programme ressemblant à...

```
100 X = 100000 : H = 10000 : X = X + H : PRINTX : GOTO 100
```

... essayez de trouver, à l'unité près, le nombre à partir duquel le MO5 ou le TO7(-70) affiche les résultats en notation "scientifique" (c'est-à-dire sous la forme : 7.2E6).

(Pour cela il vous faudra remplacer $H = 10000$ par $H = 1000$; $H = 100$; ...)

De manière analogue, découvrez le nombre maximum accepté par le MO5 ou le TO7(-70) (situé entre 1E37 et 1E38).

Les bons et les mauvais caractères

Parmi les caractères pouvant composer des chaînes acceptées par le MO5, il en est un aussi facétieux que "l'homme invisible".

C'est "l'espace" ou encore, puisqu'il faut l'appeler par son nom, le caractère " " ! (Entre les guillemets il y a un blanc, un espace que l'on obtient en frappant sur la large touche inférieure du clavier.)

Sa présence n'est détectable que si l'on affiche ce caractère sur un fond différent du fond général de l'écran.

EXPÉRIENCES

Des espaces sur fond changeant :

```
10 SCREEN 4,6,6
```

```
20 COLOR 4,I : PRINT " " : I = I + 1 : GOTO 20
```

Des espaces de plus en plus grands :

```
100 A$ = " "
```

```
110 A$ = A$ + A$ : PRINT A$ : GOTO 110
```

L'erreur de fin de programme est ici produite par le fait que, après le huitième retour à 110, A\$ dépasse 255 caractères. Cette aventure est plus facile à comprendre en exécutant :

```
200 A$ = " " : SCREEN 4,6,6 : I = 0
```

```
210 A$ = A$ + A$ : COLOR 4,I : PRINT A$ : I = I + 1 : GOTO 210
```



Attention

Ne confondez pas "l'espace" avec un acolyte tout à fait différent : "la chaîne vide". Cette chaîne là n'est constituée d'aucun caractère et

on la nomme grâce à deux guillemets qui se suivent sans encadrer quoi que ce soit : V\$ = ""

EXPÉRIENCE

10 V\$ = "" : PRINT V\$: GOTO 10

Rien ne s'affiche mais le RUN en haut d'écran disparaît après quelques temps.

Essayez donc et comparez :

10 V\$ = " " : PRINT V\$, : GOTO 10 et

10 V\$ = "" : PRINT V\$: GOTO 10

Serrer les affichages : ;

A la fin de l'exécution d'une instruction PRINT, le curseur passe en début de ligne suivante.

Pour ne pas aller à la ligne, il faut placer un **█** ou une **█** en fin d'instruction.

EXPÉRIENCE

Comparez les effets des trois instructions :

10 PRINT K : K = K + 1 : GOTO 10

10 PRINT K ; : K = K + 1 : GOTO 10

10 PRINT K , : K = K + 1 : GOTO 10

L'affichage de nombres donne lieu à des règles particulières liées à l'emploi du **█**. Ainsi le curseur avance d'un espace à la suite de l'affichage d'un nombre (les nombres sont ainsi automatiquement séparés par au moins un espace à l'affichage). De plus un caractère est prévu pour l'affichage du signe (+ ou -) et, lorsque ce signe est +, il est sous-entendu ; un espace est donc affiché au début de tout nombre positif. Finalement cela produit deux espaces entre chaque nombre d'une suite (affichée) de nombres positifs.

EXPÉRIENCE

10 PRINT 1;2; - 1; + 2; - 3

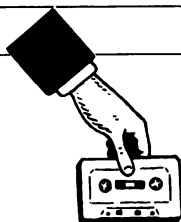
20 PRINT "1";"2";"- 1"

30 PRINT "1";1;2;"2";- 1;" + 2";- 3

CHAPITRE 4

LES OPERATIONS

Suivi de la cassette



Opérations numériques

L'emploi des "quatre" opérations que vous connaissez offre une petite difficulté due à l'obligation d'écrire des formules sur une ligne d'écran.

Ainsi :

$$\frac{A + B.C}{C - D}$$

doit s'écrire :

$$(A + B*C)/(C - D)$$

Vous voyez qu'il vous sera parfois nécessaire de rajouter des parenthèses afin que le calcul désiré s'effectue correctement. Cela dit, les priorités habituelles sur le papier restent valables (ici la multiplication $B*C$ a priorité sur l'addition $A + \dots$) .



Attention

La notation alphabétique du signe de l'opération "reste de la division" oblige à séparer par un blanc le premier terme et le signe de l'opération (lorsque le premier terme est une variable).

Ainsi le reste de la division de A par 7 s'écrit :

A M O D 7

L'écriture A MOD 7 provoque une erreur puisque les quatre lettres "A MOD" sont interprétées comme un nom de variable et la machine ne sait que faire du 7 qui suit.

Par contre l'espace entre D et le deuxième terme n'est pas nécessaire car la machine reconnaît et interprète d'abord le signe d'opération "MOD".

De même, l'écriture sur un seul niveau empêche l'emploi de la convention habituelle sur les puissances, où le signe d'opération est remplacé par une différence de niveau.

Le signe de la prise de puissance est, en BASIC, le signe \blacksquare . Ainsi on écrit :

2 \blacksquare 3 pour 2^3

3 \blacksquare (-2) pour $1/3^2$

Les opérations dont les signes sont, en BASIC, MOD et @ ne sont pas couramment utilisées dans nos collèges. Et pourtant elles sont souvent utiles dans nombre de situations pratiques (voir les compléments).

Les résultats de ces deux opérations sont le reste et le quotient de la division dite "entière". Ils sont bien visibles lorsqu'on "pose" une division et que l'on ne la poursuit pas après la virgule.

Exemple

$$\begin{array}{r|l} 365 & 7 \\ 15 & 52 \\ 1 & \end{array} \leftarrow \text{Quotient de la division entière}$$

Reste de la division entière \swarrow

On a ici : 365 MOD 7 vaut 1

365 @ 7 vaut 52

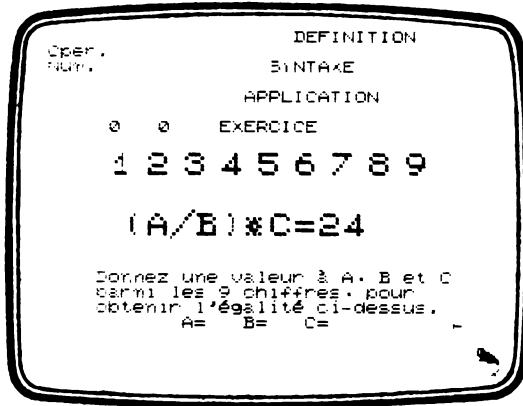


Remarque

Les trois lettres MOD viennent de l'expression "modulo".

On dit par exemple "365 modulo 7 vaut 1".

EXERCICE



Cet exercice est une sorte de “compte est bon” dans lequel on vous indique les opérations à effectuer ; à vous de trouver les bons termes pour atteindre le bon résultat.

Il vous faut frapper :

- Le premier nombre à un chiffre : A
- Le deuxième nombre à un chiffre : B
- Le troisième nombre à un chiffre : C

Attention : chacun des trois chiffres frappés doit être différent.

Par exemple, ici, pour marquer un point vous pouvez frapper :

8 2 6 ENTREE

Opération alphanumérique

La seule opération définie sur les chaînes de caractères consiste à les mettre bout à bout (à les “enchaîner” à les “concaténer”, du latin “catena” : “chaîne”).

Attention simplement à l’espace que vous désirez placer entre deux mots ! Ainsi pour écrire :

UN CHIEN

il ne faut pas commander :

A\$ = “UN” ; B\$ = “CHIEN” ; PRINT A\$ + B\$;

en effet vous obtenez ainsi :

UNCHIEN

Un bon conseil : Si vous devez faire écrire des mots les uns derrière les autres, placez un espace à la fin de chacun d'entre eux. Comme ceci :
A\$ = "UN " : B\$ = "CHIEN "

Opérations logiques

Lorsque vous programmerez couramment, vous utiliserez des conditions portant sur le résultat de comparaison. Par exemple vous voudrez exprimer que :

"Si $A > 10$, alors faire ceci, sinon faire cela".

Vous pourrez alors combiner les comparaisons du type " $A > 10$ " grâce aux opérations logiques "OU" et "ET" qui (en anglais et donc en BASIC) se notent "OR" et "AND".

Ainsi, par exemple, si vous voulez afficher "BIEN" lorsqu'une note est comprise entre 14 et 16 vous penserez :

"Si $14 < \text{NOTE} < 16$, alors afficher BIEN"

En BASIC, vous ne pourrez pas écrire un tel encadrement. Vous commanderez donc :

```
IF 14 < = NOTE AND NOTE < 16 THEN ? "BIEN"
```

Par ailleurs, si vous n'utilisez que des notes entières vous pourrez commander de manière équivalente :

```
IF NOTE = 14 OR NOTE = 15 THEN ? "BIEN"
```

Le plus étonnant (mais intéressant) c'est que de telles comparaisons soient traitées par la machine exactement comme des nombres ; ainsi vous pouvez écrire :

```
B = (14 < = NOTE) AND (NOTE < 16)
```

Les seules valeurs prises par des résultats de comparaisons sont les nombres 0 et -1 (la machine donne 0, nous pensons FAUX ; la machine donne -1, nous pensons VRAI ; la raison pour laquelle VRAI est représenté par la valeur numérique -1 et non 1 est due au codage interne des nombres !).

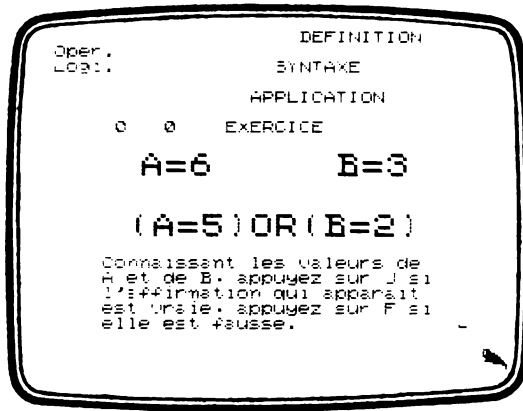
La fonction logique "NON" ("NOT" en anglais et en BASIC) échange les valeurs 0 et -1.

Ainsi :

```
C = NON (0) et C vaut -1
```

```
C = NON (-1) et C vaut 0
```

EXERCICE



Pour marquer un point il vous faut ici frapper sur F. En effet, l'affirmation $(A = 5 \text{ ou } B = 2)$ est fausse puisque ni A ne vaut 5, ni B ne vaut 2.



La prise de puissance

Dans l'écriture $A \blacksquare B$, lorsque l'exposant B est un entier, le premier terme A peut être quelconque (positif ou négatif).

$(-1) \blacksquare 4$ vaut 1

$\emptyset \blacksquare 1$ vaut \emptyset .

Mais l'exposant B peut aussi être négatif :

L'écriture $3 \blacksquare (-4)$ équivaut à $1/(3 \blacksquare 4)$

Le premier terme A ne peut alors être nul.

Mieux encore, l'exposant B peut être un nombre réel quelconque.

Pour que l'expression écrite ait un sens il faut alors que le premier terme A soit positif.

Ainsi :

$3 \blacksquare (1/2)$ vaut $\sqrt{3}$

Et, pour ceux qui ont fait des mathématiques :

$3 \blacksquare (2.53)$ vaut $e^{2,53 \text{Log}3}$

Plus généralement, l'écriture $A \blacksquare B$ équivaut à l'écriture :

EXP (B*LOG(A)).

La pratique du modulo

Vous savez, par exemple, que votre écran peut contenir 25 lignes de 40 caractères.

Si l'écriture d'un texte a commencé en haut à gauche, sauriez-vous dire sur quelle ligne et sur quelle colonne est écrit le 365^{ème} caractère de ce texte.

Si l'on décide que le 1^{er} caractère porte le numéro \emptyset , alors le 365^{ème} caractère porte le numéro 364.

Le résultat est simple à obtenir :

$$364 @ 40 = 9.$$

Ce caractère est sur la ligne numéro 9...

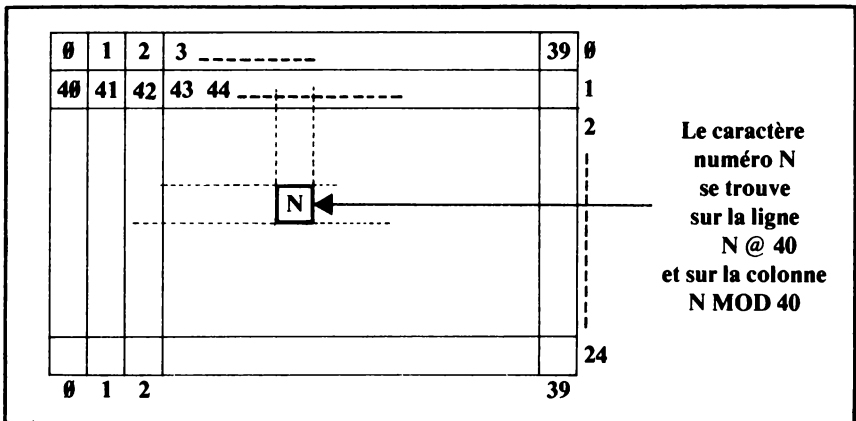
$$364 \text{ MOD } 40 = 4$$

... et c'est le caractère numéro 4 de cette ligne.

Attention

Pour que les choses marchent aussi bien il a fallu décider que la première ligne porterait le numéro zéro, et non pas le numéro un (de même pour les colonnes).

Ainsi la conclusion du calcul précédent est que le 365^{ème} caractère se trouve sur la 5^{ème} colonne de la 10^{ème} ligne.



Prenez donc une bonne habitude dans le contexte informatique :

Pour compter une suite de choses, il est commode de commencer à compter à partir de 0 :

La 1^{ère} chose de la suite porte le numéro 0.

La 2^{ème} chose de la suite porte le numéro 1.

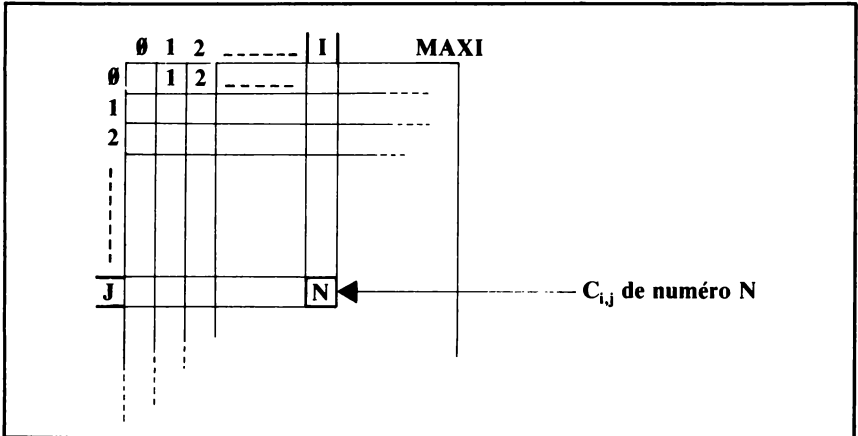
... et ainsi de suite...

La (N + 1)^{ème} chose de la suite porte le numéro N.

Vous verrez que les calculs se simplifieront grâce à cette convention pratique. Alors, plus généralement que des caractères sur votre écran, vous pouvez retenir ceci :

Si des choses sont rangées dans un tableau et repérées par deux indices i et j (i variant de \emptyset à MAXI et j variant à partir de \emptyset), alors :

- la chose notée $C_{i,j}$ porte le numéro N avec $J = N @ \text{MAXI}$ et $I = N \text{ MOD } \text{MAXI}$.



Des chiffres et des nombres

Imaginons que vous connaissiez les chiffres d'un nombre de trois chiffres : C , le chiffre des centaines ; D le chiffre des dizaines et U le chiffre des unités.

Par exemple, si C vaut 4, D vaut 2 et U vaut 1, vous croyez écrire le nombre 421 en commandant :

`PRINT C;D;U`

... et vous voyez apparaître à l'écran ceci :

4 2 1

Des espaces intempestifs se sont glissés dans l'écriture.

Pourquoi ?

- D'abord parce que la machine réserve toujours une place pour le signe du nombre... et qu'elle n'indique pas ce signe lorsqu'il s'agit de "+".

- Ensuite parce que la machine sépare l'affichage d'un nombre et d'un autre par un espace (ce qui est bien souvent très pratique pour la lisibilité des résultats).

EXPÉRIENCES

$A = 1 : B = -2 : C = -3 : ?A;B;C$

$A = -1 : B = 2 : C = -3 : ?A;B;C$

$A = -1 : B = -2 : C = -3 : ?A;B;C$

Pour faire “coller” les chiffres les uns aux autres, il faut les afficher comme des chaînes de caractères.

$A\$ = "4" : B\$ = "2" : C\$ = "1" : ?A\$ + B\$ + C\$$

Évidemment, la plupart du temps, les chiffres sont des variables, appelées par exemple C,D,U... Il faut alors utiliser une fonction qui transforme les nombres en chaîne de caractères. Cette fonction est la fonction STR\$. Par exemple si C vaut 4, alors STR\$(C) vaut “4” (voir plus loin).

La priorité des opérations

Dans une expression algébrique :

— S’il n’y a que des additions et des soustractions, les opérations se font de la gauche vers la droite.

— S’il n’y a que des multiplications et des divisions, les opérations se font de la gauche vers la droite.

— S’il n’y a que des prises de puissance, elles se font de la gauche vers la droite.

EXPÉRIENCES

Essayez de prévoir à l’avance le résultat des commandes suivantes :

$A = 24 : B = 12 : C = 6 : D = 4 : E = 2 : ?A - B - C + D - E$

$?A/B*C/D*E$

$?A/B/C/D/E$

$?A/B/C*D*E$

$?A*B/C*D/E$

$?2 \blacksquare 3 \blacksquare 4$

$?4 \blacksquare 3 \blacksquare 2$

— S'il y a plusieurs signes d'opérations et si des parenthèses ne précisent pas leur ordre, les opérations se font dans l'ordre suivant : d'abord les prises de puissance, ensuite les multiplications et les divisions, puis les additions et les soustractions.

EXPÉRIENCES

$$A = 1 : B = 2 : C = 3 : D = 4$$

$$?A + B/C + D$$

$$?A * B + C * D$$

$$?A/B \blacksquare C + D$$

$$?A + B \blacksquare C/D$$

$$?A - B * C - D$$

Les comparaisons enchaînées

Gags

Essayez de ne pas vous perdre dans l'interprétation de ces suites de commandes :

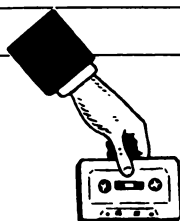
$A = \emptyset$ $B = A$ $C = B$ $?A = B = C$	$A = -1$ $B = A$ $C = B$ $?A = B = C$
$A = \emptyset$ $B = -1$ $C = \emptyset$ $?A = B = C$	$A = \emptyset$ $B = -1$ $C = \emptyset$ $?A = C = B$

Aviez-vous prédit les résultats ?

CHAPITRE 5

NOTIONS DE PROGRAMMATION

Suivi de la cassette



Ce chapitre précise les éléments déjà exposés dans la deuxième partie de l'introduction.

Il y aura donc peu d'exercices et pas du tout de compléments.

Modes

Normalement, un ordinateur est fait pour être utilisé en "mode-programme". Vous l'utiliserez sûrement en "mode-direct" dans deux cas au moins :

— Lorsque vous voulez connaître le résultat d'un petit calcul ; par exemple pour calculer le volume d'une sphère de rayon donné, vous écrirez :

$PI = 3.1416$: $R = 6,2$: ? $4 * PI * R * R * R / 3$


(Vous verrez ainsi qu'un ballon de rayon 6,2 cm n'est pas loin de contenir 1 litre.)

— Lorsque vous voulez impressionner vos amis avec une suite de commandes dont l'énoncé ne nécessite pas plus de 255 caractères (6 lignes et quelques caractères).

En voici un exemple, anticipant sur une instruction du chapitre 10 :

```
SCREEN 1,0,0 : FOR I = 1 TO 7 : BOXF
```

```
(10*I,10*I) - (200 - 10*I,200 - 10*I) : NEXT I
```


 **Attention**

Évidemment le désavantage d'une suite d'instructions en mode direct c'est que, lorsque son énoncé à disparu de l'écran, il n'est plus possible de l'exécuter.

Comme un petit numéro en début de ligne ne coûte pas grand chose, il vaut souvent mieux le glisser pour passer en mode-programme et l'instruction est mémorisée (d'autant plus qu'il est facile de l'effacer si elle devient gênante).

Ligne

 **Attention**

Une "ligne de programme" peut occuper plus d'une ligne "physique" visible sur l'écran. Elle peut même en occuper sept !

Une ligne de programme se présente ainsi :

```
numéro une instruction : une instruction :  
..... une instruction  
ENTRÉE
```

 **Remarque**

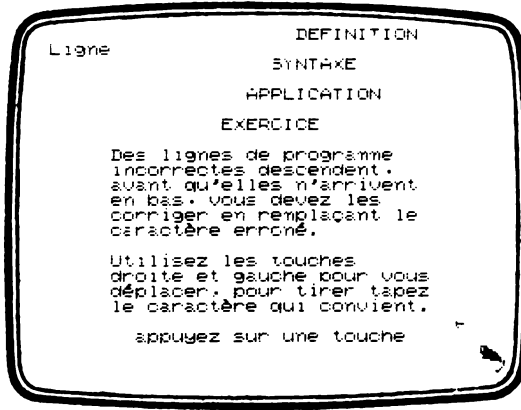
1. Lorsque vous frappez un caractère alors que le curseur se trouve sur le dernier caractère d'une ligne, le curseur se placera ensuite au début de la ligne suivante et un lien sera créé entre les deux lignes.

2. Lorsque vous frappez sur **ENTRÉE**, le curseur peut se trouver non seulement en haut de ligne mais quelque part n'importe où (cela arrive si vous avez voulu corriger quelque erreur de frappe, par exemple). L'ordinateur repère alors la position du curseur et interprète la ligne du curseur en même temps que toutes les lignes qui ont été "liées" à elle.

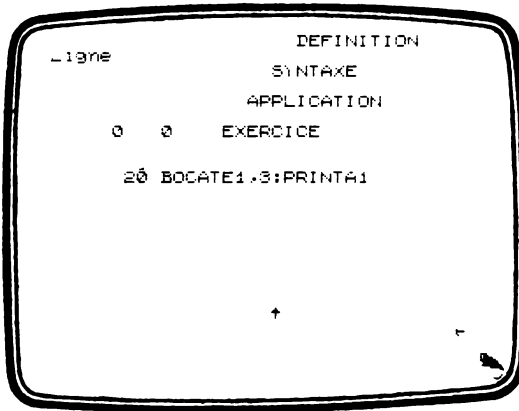
Pour effacer une instruction, de numéro 37 par exemple, il suffit de frapper le numéro (**3 7**) suivi de **ENTRÉE**.

Par contre, pour regarder une instruction, il faut frapper **L I S T** suivi de son numéro (**3 7**) puis de **ENTRÉE**.

EXERCICE



Appuyez sur une touche pour commencer l'exercice.



Attention, la ligne d'instruction descend assez vite ! Ici il vous faut corriger la première lettre de "BOCATE" qui devrait être un "L".

Il vous faut donc frapper neuf fois sur **←** pour déplacer votre "canon à caractères" puis frapper sur **L** avant que la ligne n'atteigne le bas de la boîte colorée.

RUN END

“RUN” signifie “courir” et “END” signifie “fin”.

Lorsqu’on commande l’exécution d’un programme, tous les noms de variables contiennent **0** (pour les nombres) et **” ”** pour les chaînes. Par exemple, avec le programme

```
1 PRINT A,Z, “OUI” ; A$ ; “NON”
```

La commande RUN produit l’affichage :

```
  0.....0.....OUI NON
```

La possibilité de faire commencer l’exécution d’un programme à une ligne quelconque vous permet de garder en mémoire plusieurs programmes différents. Par exemple :

```
 1 _____  
  : Programme de calcul d’une moyenne  
  :  
  :  
99 END _____
```

```
100 _____  
  : Programme de calcul du jour de la semaine  
  :  
  :  
199 END _____
```

```
200 _____  
  : Programme de mise au pluriel des noms  
  :  
  :  
299 END _____
```

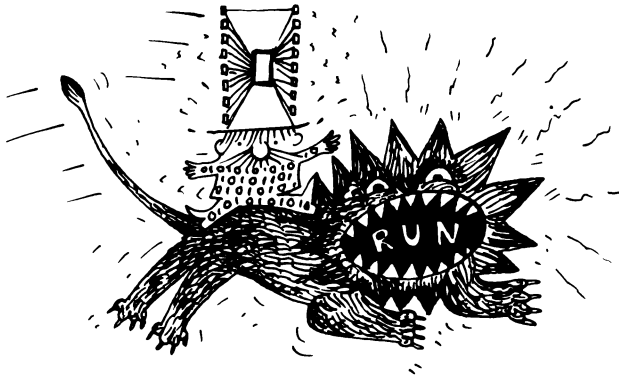
Pour calculer le jour de la semaine, vous pouvez alors commander :

```
RUN 100 ENTREE
```



Attention

N’oubliez pas de terminer vos programmes par l’instruction END. Cela n’est pas obligatoire lorsqu’il n’y a qu’un seul programme en mémoire mais cela est une bonne habitude.



STOP CONT

Vous savez déjà que l'arrêt intempestif d'un programme en exécution peut être commandé en appuyant sur les touches **CNT - C**.

Mais vous pouvez aussi prévoir de placer une instruction STOP dans votre programme. Cela vous permettra de continuer l'exécution quand vous le désirerez en commandant : **CONT ENTREE**.

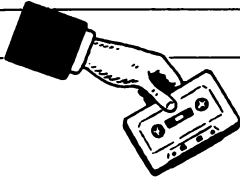


Remarque

Lorsque vous avez interrompu l'exécution d'un programme (par STOP ou CNT-C), les variables ne sont pas remises à zéro. Vous pouvez donc en faire afficher les valeurs ; cela peut vous être très utile si vous ne comprenez pas le fonctionnement d'un programme ou s'il ne fait pas exactement ce que vous voulez. En glissant quelques instructions STOP bien placées, vous pouvez ainsi suivre la valeur des variables qui vous intéressent à tout moment de l'exécution.

CHAPITRE 6

Aides à la programmation



Suivi de la cassette

Exceptionnellement (mais un peu comme le précédent) ce chapitre ne comporte pas d'exercices. Il s'agit en effet d'instructions très simples qu'il faut savoir utiliser. Pour vous y habituer nous vous proposerons, dans ce livret, quelques activités et expériences intéressantes.

Les instructions étudiées dans ce chapitre auraient peu d'intérêt s'il était possible d'écrire du premier coup des programmes qui marchent parfaitement.

L'erreur est (heureusement ?) humaine et il nous faut pouvoir corriger, supprimer, ou suivre à la trace les instructions que nous avons nous-mêmes données.

LIST

“Lister” un programme, c'est afficher la liste des instructions qui le compose.

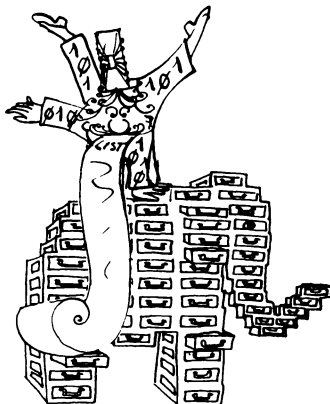
La commande LIST n'a qu'un petit défaut : les instructions défilent très vite sur l'écran ; et si votre programme dépasse les 25 lignes, le début disparaît en haut pendant que la suite défile vers le bas.

Pour observer une certaine partie d'un programme, vous avez alors deux possibilités :

— Commandez la liste complète et concentrez-vous sur le défilement rapide des instructions en appuyant sur **STOP** de temps en temps ; si les instructions qui vous intéressent sont plus “loin” dans le programme, appuyez sur **■** pour continuer le défilement... et ceci jusqu'à ce que soient affichées les instructions qui vous intéressent. Appuyez alors sur **CNT - C**.

— Commandez la liste d'une partie du programme. Par exemple LIST 210-250 fait afficher les instructions numérotées entre 210 et 250 (inclus).

Pour corriger ou modifier une instruction, affichez-la d'abord à l'écran. Ainsi l'instruction LIST 36 affiche-la seule instruction 36. Le curseur plein écran vous permet alors de la modifier selon votre volonté.



DELETE

“DELETE” signifie “effacer”.

Heureusement la commande DELETE ne détruit pas votre programme ; elle provoque une erreur et attire ainsi votre attention sur le fait que vous semblez vouloir effacer l'œuvre entière que vous aviez entreprise.

REM

“REM” est l'abréviation de “REMarque”.

Il s'agit de l'instruction la plus apparemment inutile de toutes les instructions.

En effet, l'apparition de ces trois lettres provoque un désintéressement complet de la machine pour tout ce qui est écrit ensuite sur la même ligne de programme : les caractères qui suivent ne sont tout simplement pas interprétés !

Vous pourriez trouver cette instruction un peu "idiote". Mais justement elle est très importante dans au moins deux cas :

— Si vous écrivez un programme composé de nombreux ensembles d'instructions se suivant ou s'imbriquant les uns les autres, vous avez intérêt à titrer ces ensembles d'instructions par des instructions REM qui vous aideront à vous y retrouver.

(Il y a même des programmeurs qui commencent l'écriture d'un programme par la simple succession de tels titres puis qui détaillent ensuite les instructions de chaque ensemble !)

— Si vous abandonnez un programme pendant quelques jours, vous ne pourrez sûrement pas vous y reconnaître pour le continuer ou le modifier si vous n'avez pas ponctué vos instructions de remarques concernant les variables ou les ensembles significatifs d'instructions.

(La même nécessité se retrouve dans le cas où un même programme est élaboré ou modifié par une équipe de programmeurs. Même si cette équipe est très réduite, les remarques apportent un "sens" et donc une meilleure compréhension qui permet un travail plus efficace et rapide.)

TRON TROFF

"TRON" est l'abréviation de "TRACE ON" signifiant "mode-trace en action".

Le "mode-trace" consiste, pour la machine, à afficher la trace de tout ce qu'elle fait. Précisément elle affiche les numéros des lignes d'instructions qu'elle exécute !

Évidemment, elle affiche ce numéro à l'endroit où le curseur se trouve lors de l'exécution de l'instruction. (Dans le cas de programmes utilisant le graphique, cela fait souvent un peu "fouillis".)

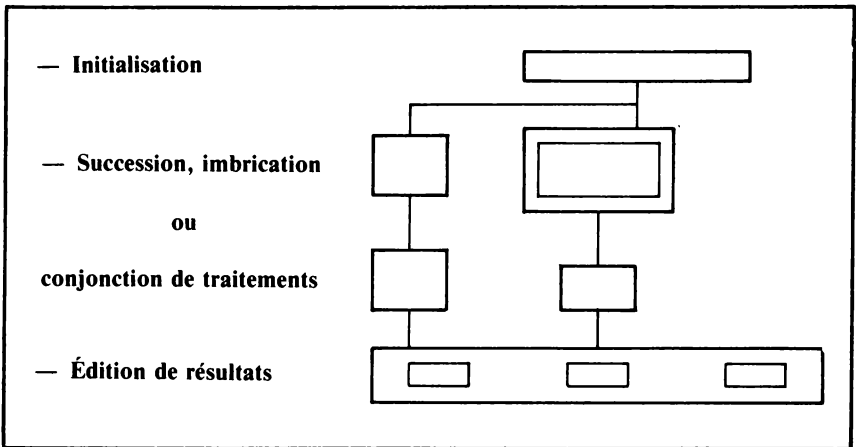
"TROFF" est l'abréviation de "TRACE OFF" signifiant "fin d'action du mode-trace".



Commentez vos programmes

Lorsque vous écrirez de longs programmes il sera important pour vous de bien faire apparaître les ensembles d'instructions, correspondant à des traitements constituant une unité.

Souvent la structure de votre programme se présentera comme ceci :



Un bon moyen pour bien délimiter les blocs d'instructions consiste à utiliser des lignes de *, de -, de . ou de #, ... dans des instructions REM.

Voici, en exemple, un tout petit programme comptant de P en P à partir d'un nombre N.

(Vous pouvez jouer, pendant l'exécution de ce programme, en essayant de l'arrêter par STOP le plus près de 1000 par exemple.)



Attention

R E M =

L'apostrophe est plus commode et plus discrète que les trois lettres.


```

10 ***** INITIALISATION *****
20 N = 150
30 P = 36
40 '*****
50 ***** BOUCLE PRINCIPALE *****
60 LOCATE 10,10
70 PRINT N
80 N = N + P
90 GOTO 50
100 '*****
110 END

```

Les faux commentaires

Il arrive que l'on se pose des questions sur l'opportunité ou sur l'effet d'une certaine instruction.

Par exemple dans le programme du paragraphe précédent, on voudrait voir (avec les yeux) l'effet produit par l'instruction 60. Et si on la supprimait, qu'est-ce que cela donnerait à l'exécution ?

Pour cela, il est pratique de placer momentanément une apostrophe après le numéro de l'instruction. Comme ceci :

```

LIST 60  ENTREE
↑ → → → → ENTREE
RUN  ENTREE

```

Tout se passe alors comme si l'instruction 60 avait été supprimée (puisqu'elle est traitée comme une ligne de commentaires).

Mais ce qui est pratique, c'est que l'on peut rétablir cette instruction en enlevant simplement l'apostrophe (au lieu d'avoir à réécrire toute l'instruction) comme ceci :

```

LIST 60  ENTREE
On voit alors apparaître :
60 ' LOCATE 10,10
On peut alors frapper :
↑ → → → ENTREE

```

... et on est revenu au programme initial après avoir essayé une petite variante.

Si vous n'êtes pas encore très sûr de bien comprendre ce qui se passe lors de l'exécution du programme précédent, nous vous conseillons de suivre les instructions "à la trace".

EXPÉRIENCE

TRON **ENTREE**

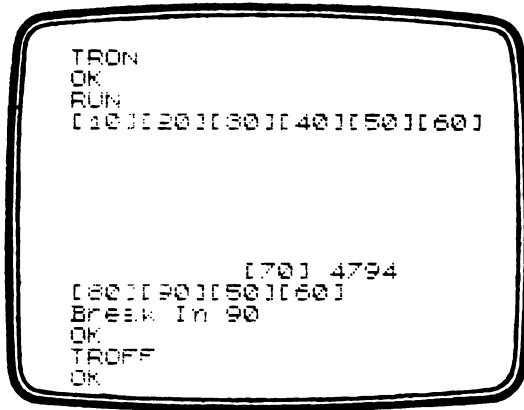
RUN **ENTREE**

Après quelques instants de contemplation réfléchie appuyez sur :

CNT - C

TROFF **ENTREE**

Vous avez dû voir l'écran suivant s'écrire ligne après ligne :



```
TRON
OK
RUN
[10][20][30][40][50][60]

                                [70] 4794
[80][90][50][60]
Break In 90
OK
TROFF
OK
```

Un léger clignotement des numéros 60, 70, 80, 90 et 50 atteste que ces instructions se répètent indéfiniment.

Vous pouvez ainsi vérifier que :

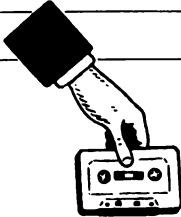
— L'instruction 60 positionne le curseur à la 10^{ème} colonne de la 10^{ème} ligne.

— Après l'instruction 70 et l'impression de la valeur de N, la machine est allée à la ligne.

— Après l'instruction 90, la machine effectue l'instruction 50 avant de se repositionner en (10,10).

CHAPITRE 7

Caractères utilisateurs



Suivi de la cassette

Vous allez maintenant apprendre non seulement à dessiner la forme d'une voiture mais aussi à donner un nom à cette forme, de manière à pouvoir la reproduire sur l'écran au moment, à l'endroit et autant de fois que vous voulez.

CLEAR

Avant de définir votre voiture, il y a une petite contrainte due à l'organisation interne de votre ordinateur.

En effet toutes les formes de caractères comme "A", "7" ou "#" sont définies à l'intérieur de la machine dans des mémoires spéciales (et protégées !).

Définir une nouvelle forme c'est véritablement "créer" un nouveau caractère qu'il faudra donc aussi ranger dans une mémoire spéciale (à côté, pourrait-on dire, des caractères déjà "connus" par la machine).

Il est alors nécessaire de prévenir la machine qu'elle aura à mémoriser de nouveaux caractères et il faut lui indiquer le nombre de nouveaux caractères que vous allez lui définir, afin qu'elle réserve la place du codage de leur forme.

L'instruction permettant de réserver les mémoires nécessaires au codage de six nouveaux caractères est ainsi :

CLEAR , , 6

“CLEAR” signifie “nettoyer” (ici “nettoyer la place” pour y mettre ensuite quelque chose).

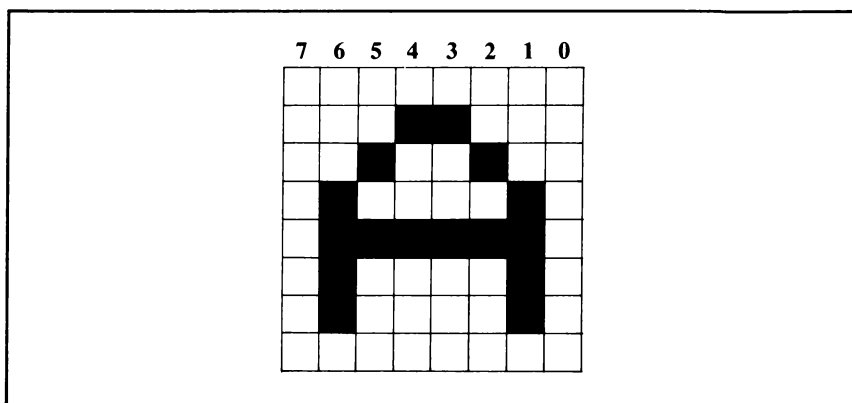
Les deux virgules suivant le R ne doivent pas vous troubler ; elles sont là pour indiquer que vous voulez réserver des places pour le 3^{ème} type de réservation possible dans votre ordinateur.

Il y a, en effet, deux autres types de réservation dont vous pourriez indiquer les caractéristiques avant la 1^{ère} virgule et entre la 1^{ère} et la 2^{ème} virgule ; mais ces réservations-là ne vous seront aujourd’hui d’aucune utilité !

Un conseil : Sauf pour des raisons d’économie, n’hésitez pas à réserver toujours plus de places qu’il ne vous semble nécessaire au début.

DEF GR\$

Afin de mieux comprendre comment est codé un caractère, examinons le dessin du “A” sur l’écran :



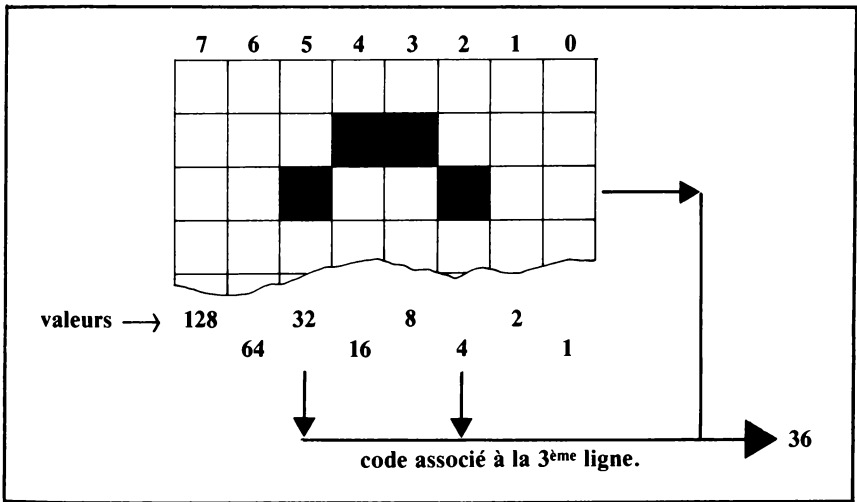
Pour transmettre à quelqu’un la forme de ce A vous pourriez agir ainsi :

- Décider de travailler sur une grille 8×8.
- Numéroté les colonnes de 0 à 7 comme indiqué sur le dessin.
- Décrire successivement chaque ligne depuis celle du haut jusqu’à celle du bas, comme ceci :

- 1^{ère} ligne : rien
- 2^{ème} ligne : col3, col4
- 3^{ème} ligne : col2, col5
- 4^{ème} ligne : col1, col6
- 5^{ème} ligne : col1, col2, col3, col4, col5, col6
- 6^{ème} ligne : col1, col6
- 7^{ème} ligne : col1, col6
- 8^{ème} ligne : rien

Vous pouvez abréger ce message ainsi :
rien, 3 et 4, 2 et 5, 1 et 6, 1 et 2 et 3 et 4 et 5 et 6, 1 et 6, 1 et 6, rien.
C'est là que la magie des mathématiques intervient !

Plutôt que de numéroter les colonnes de 0 à 7 et d'indiquer les colonnes marquées, il vaut mieux attribuer une valeur à chaque colonne et ajouter ces valeurs. En utilisant comme valeurs les puissances de deux ($2^0, 2^1, 2^2, \dots, 2^7$) on peut toujours retrouver les colonnes marquées à partir de la somme des valeurs.



C'est ainsi que le "A" sera codé par une suite de 8 nombres :
 $0, 8 + 16, 4 + 32, 2 + 64, 2 + 4 + 8 + 16 + 32 + 64, 2 + 64, 2 + 64, 0$

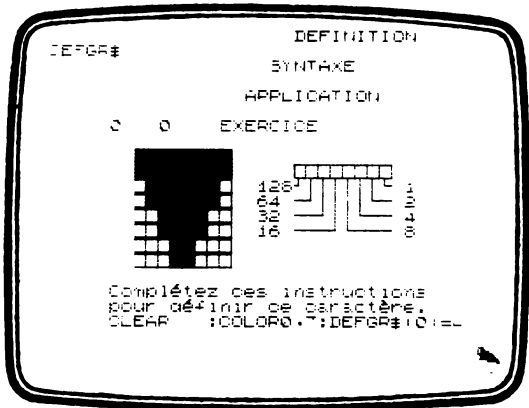
C'est-à-dire, après avoir effectué les additions :
 $0, 24, 36, 66, 126, 66, 66, 0$

Si le "A" n'était pas déjà défini dans la machine, vous pourriez le faire en commandant :

```
DEF GR$ (1) = 0, 24, 36, 66, 126, 66, 66, 0
```

Et voilà comment vous pouvez définir une forme (dans une grille 8 × 8) en donnant une liste de 8 nombres (compris entre 0 et 255), chacun des nombres constituant le code d'une ligne et indiquant les points "marqués" sur cette ligne !

EXERCICE



Cet exercice vous entraîne au calcul des "codes-ligne", correspondant à des configurations de points donnés.

Ici vous devriez écrire "1" après CLEAR, puis █, puis 255, 255, 126, 126, 60, 60, 24, 24 **ENTREE**

GR\$

Il n'y aurait évidemment aucun intérêt à la définition de caractères s'il n'existait pas une instruction permettant de les utiliser et de les afficher.

Les caractères créés doivent être numérotés à partir de 0 (et non à partir de 1).

Attention

Si vous définissez 6 caractères, le dernier portera le numéro 5 :

DEF GR\$ (0) =	DEF GR\$ (3) =
DEF GR\$ (1) =	DEF GR\$ (4) =
DEF GR\$ (2) =	DEF GR\$ (5) =

Le nom du caractère défini par l'instruction DEF GR\$ (3) est GR\$ (3).



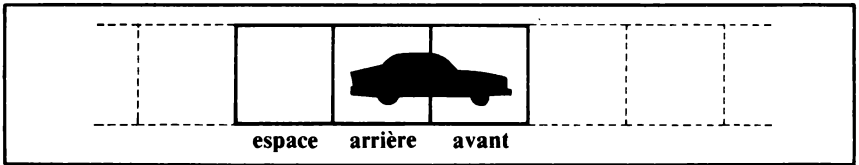
Remarque

“DEF GR\$” est l’abréviation de “définir” et de “graphique” ; le \$ indique que votre création sera traitée comme une donnée de type alphanumérique.

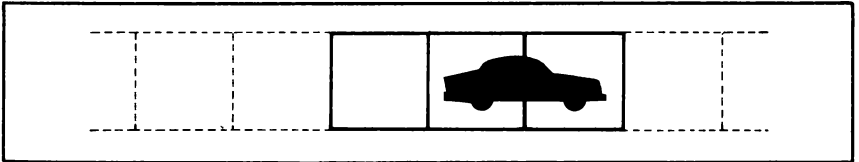
A partir du moment où votre nouveau caractère porte un nom, vous pouvez lui faire subir des opérations (le concaténer à d’autres caractères par exemple) ou bien l’afficher (seul ou avec d’autres)...

Dans l’application donnée, l’avance de la voiture est obtenue par un procédé astucieux mais classique :

— On affiche la voiture mais en ajoutant un espace après son arrière. Le dessin affiché est donc constitué de 3 caractères :

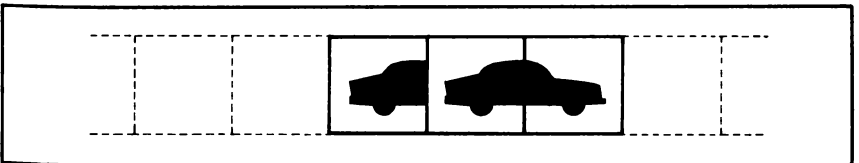


— On fait avancer le curseur d’une position et on affiche les 3 mêmes caractères :



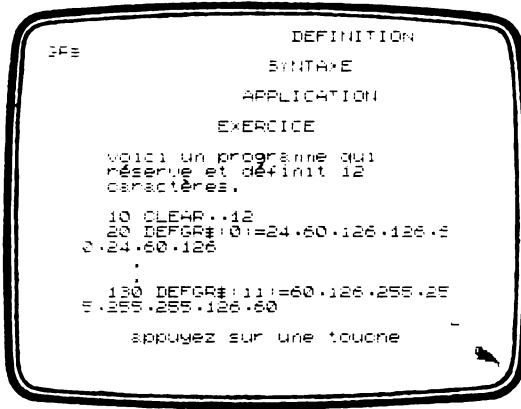
... et l’espace à l’arrière de la voiture remplace alors l’arrière de la 1^{ère} voiture dessinée.

Si vous n’avez pas placé cet espace, on aurait eu l’affichage suivant :

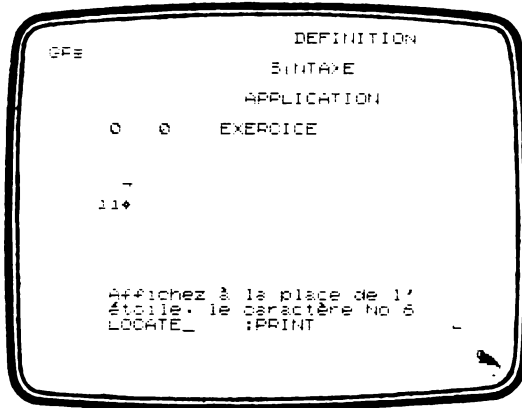


Cette méthode est dite “de la queue du renard” (le petit malin qui efface avec sa queue les traces de pas qu’il laisse dans la neige).

EXERCICE



Après l'affichage d'une première page, appuyez sur une touche quelconque.



Ici, il vous faut répondre :

7 . 1 1 → G R S (6) ENTREE

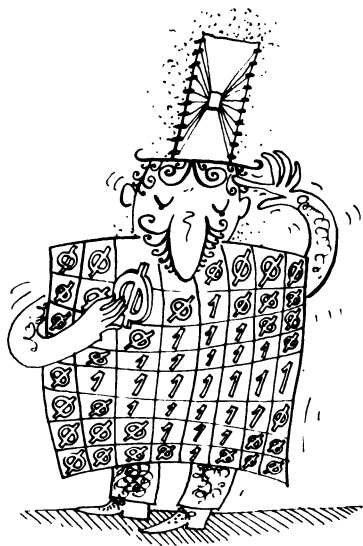


Compléments, activités et expériences

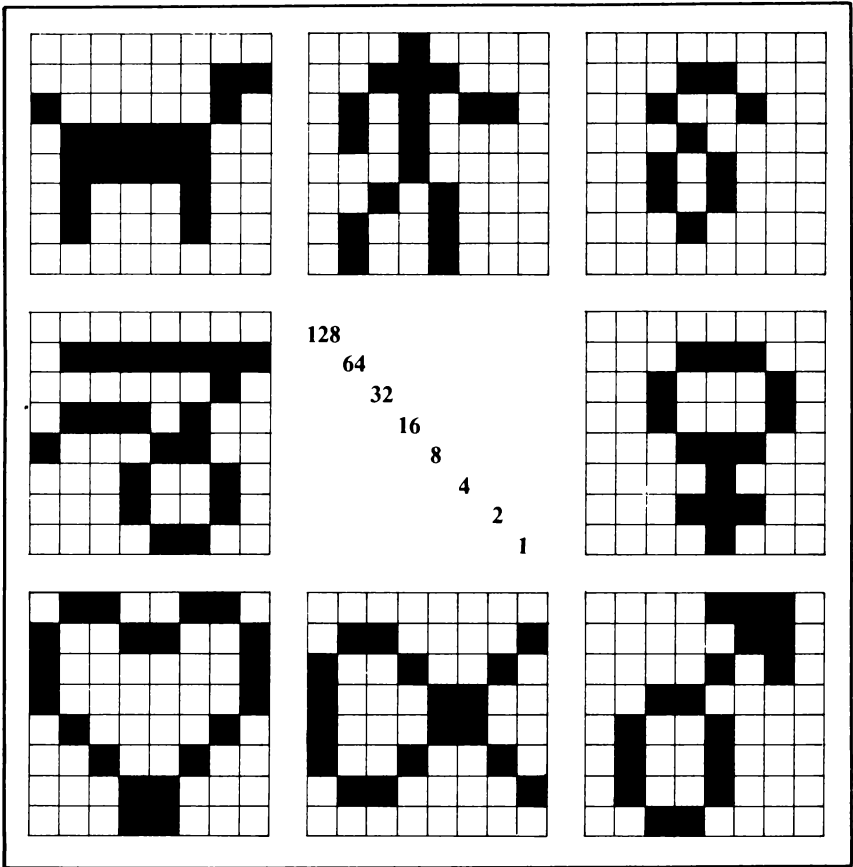
Des lettres grecques, des animaux, des symboles

Nous vous offrons simultanément un jeu et quelques codages de formes.

Voici, en effet, huit dessins et huit séries de code (instructions 20 à 90).



C'est à vous de retrouver les codes correspondant à chacun des dessins et de les vérifier par le programme suivant :



10 CLEAR , , 8

20 DEF GR\$(0) = 0, 28, 34, 34, 28, 8, 28, 8

30 DEF GR\$(1) = 0, 3, 130, 124, 124, 68, 68, 0

40 DEF GR\$(2) = 0, 97, 146, 140, 140, 146, 97, 0

50 DEF GR\$(3) = 0, 24, 36, 16, 56, 56, 16, 0

60 DEF GR\$(4) = 0, 127, 2, 116, 140, 18, 18, 12

70 DEF GR\$(5) = 14, 6, 10, 48, 72, 72, 72, 48

80 DEF GR\$(6) = 16, 56, 86, 80, 16, 40, 72, 72

90 DEF GR\$(7) = 102, 153, 129, 129, 66, 36, 24, 24

100 I = 0

110 PRINT I, GR\$(I)

120 PRINT

130 I = I + 1

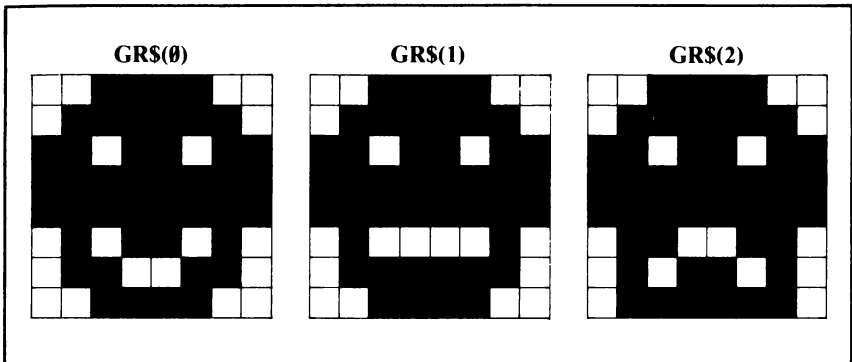
140 GOTO 110

RUN... et c'est parti !

Qui rit vendredi, dimanche pleurera

L'affichage successif au même endroit permet de donner l'impression d'une variation continue.

Par exemple, on peut superposer un visage souriant, un visage neutre et un visage triste, comme ceci :



ACTIVITÉS

Écrivez le morceau de programme définissant les trois visages numérosés 0, 1 et 2.

EXPÉRIENCES

Le programme suivant simule alors les états successifs d'un cyclothymique excessif :

```
10 ATTRB 1,1 : SCREEN 1,6,6
20 LOCATE 10,10 : PRINT GR$ (0)
30 LOCATE 10,10 : PRINT GR$ (1)
40 LOCATE 10,10 : PRINT GR$ (2)
50 GOTO 20
```

Un dessin animé

En combinant les effets de la voiture qui avance et du visage qui change d'expression, on dispose des principes de fabrication d'un véritable dessin animé.

Par exemple pour fabriquer un chien qui marche on utilisera trois images successives affichées alternativement.

Voici la structure du programme qu'il vous reste à écrire :

```
10 'Définition des trois positions du chien
11 CLEAR ,, 3
12 DEF GR$ (0) = (chien pattes droites)
13 DEF GR$ (1) = (chien pattes en avant)
14 DEF GR$ (2) = (chien pattes en arrière)
19 '
20 A$ = " " + GR$ (0)
21 B$ = " " + GR$ (1)
22 C$ = " " + GR$ (2)
30 'Initialisation de la position
31 P = 0
40 'Avance du chien
50 LOCATE P,10 : ? A$
60 LOCATE P + 1,10 : ? B$
70 LOCATE P + 2,10 : ? C$
80 P = P + 3
90 GOTO 40
```

Décodage d'une forme

Le problème que vous avez le droit de vous poser est le suivant :

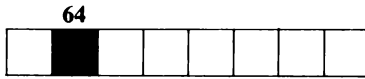
“Étant donné les codes des huit lignes d'une forme, comment retrouver cette forme.”

Pour cela, il vous suffit de savoir...

“étant donné le code d'une ligne, comment retrouver les points marqués sur cette ligne.”

Voici, par exemple, comment retrouver la ligne correspondant au nombre 101 :

— 101 étant plus petit que 128, la colonne “128” n’est pas marquée ; par contre 64 est marqué

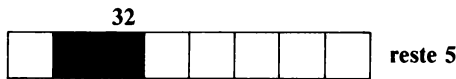


Règle générale : chercher le point marqué le plus à gauche (c’est-à-dire la puissance de deux la plus grande, contenue dans le nombre).

— Il reste donc à trouver les points marqués correspondant à $101 - 64$ c’est-à-dire 37.

Règle générale : soustraire la valeur du point trouvé et recommencer à appliquer la première règle.

— 37 contient 32



— 5 contient 4



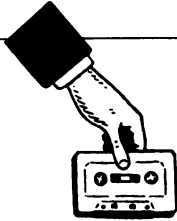
— 1 contient 1



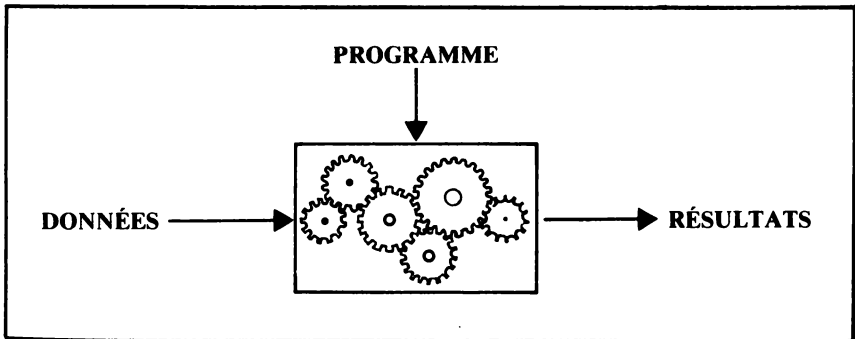
CHAPITRE 8

Entrée des données

Suivi de la cassette



Le mot “données” englobe, en informatique tout ce qui est “donné” à l’ordinateur en dehors des commandes. Ce peut donc être des nombres, des textes ou de simples caractères. (Ce peut être aussi des impulsions électriques ou des touches de crayon, mais il n’en sera pas question dans ce chapitre.)



En ce qui concerne les données alphanumériques, il y a deux possibilités :

— Ou bien on entre simultanément en mémoire le programme et les données.

— Ou bien on entre le programme d’abord, puis, au moment de son exécution, l’utilisateur est interrogé et doit entrer les données (qui lui sont alors demandées par le programme).

L'avantage de la première possibilité est que, les données étant en mémoire en début d'exécution, le programme s'exécute sans s'arrêter et sans intervention de l'utilisateur. Cependant, puisqu'il n'y a pas d'entrée d'informations extérieures il réalise toujours les mêmes actions (avec seulement une possible intervention du hasard).

L'avantage de la deuxième manière est donc de fournir à l'ordinateur des données différentes à chaque exécution ; le programme est bien une espèce de machine prête à fonctionner avec les ingrédients que voudra bien lui donner l'utilisateur, et le travail fourni dépendra de ces données.

Évidemment l'entrée de données par l'utilisateur ralentit le fonctionnement du programme et oblige l'utilisateur à frapper sur des touches.

Il y a donc un choix à faire pour le programmeur :

... les données qui feront partie du programme (ce sont celles que de nombreux utilisateurs, sinon tous, entreraient identiques — leur modification nécessite la correction de lignes de programmes).

Ces données utilisent les instructions READ et DATA.

... et les données que rentrera l'utilisateur lors de l'exécution (ce sont celles, peu nombreuses, qui seront spécifiques à chaque utilisateur).

Ces données utilisent les instructions INPUT ou INKEY\$.

INPUT

“INPUT” signifie “entrée”.

Examinons en détail ce que fait la machine après la demande d'exécution d'un programme contenant cette instruction.

Voici ce qui a été frappé :

```
10 INPUT "PRIX" ; P  ENTREE
20 INPUT "QUANTITE" ; Q  ENTREE
30 PRINT "COUT:" ; P*Q  ENTREE
RUN  ENTREE
```

La machine interprète la première instruction de programme (ici l'instruction numéro 10) :



INPUT

Bon, on va m'entrer des données, voyons plus loin de quoi il s'agit...

“ Ah ! ça commence par des caractères que je dois afficher sur l'écran ; au boulot !... ”



P J'affiche : P

R J'affiche : R

I J'affiche : I

X J'affiche : X

“ Ah ! j'arrête d'afficher, voyons la suite... ”

; voilà le séparateur ! merci, je continue...

P Ce doit être le début du nom de la variable dont on doit me donner la valeur...



ENTREE

Tiens ! c'est déjà fini, la variable s'appelle donc P ; c'est une variable numérique. Il faut que je lui en demande la valeur : je “lui” affiche un point d'interrogation et j'attends “ses” ordres.



PRIX ?

Mais, qu'est-ce qu'“il” fait là-haut, il est bien long !

L'utilisateur doit alors, en effet, frapper des chiffres ; imaginons qu'il frappe ceci :

1 2 F ENTREE



Ah ! "il" a frappé sur ENTREE ; voyons un peu la valeur qu'"il" a donnée à P. Je recopie : $P = 12$ F. Mais qu'est-ce que c'est que ce F ! Il était prévu de me donner un nombre c'est-à-dire une suite de chiffres ; je ne sais quoi faire de cette lettre. Je vais lui envoyer un message d'erreur et attendre une bonne suite de chiffres.

Et la machine affiche :

REDO

?

Si l'utilisateur a lu le mode d'emploi, il frappera alors sûrement :

1 2 ENTREE



Bon ! cette fois-ci je pose $P = 12$ et j'interprète l'instruction suivante...

Et la machine affichera...

QUANTITE ?

... puis attendra la valeur de Q.

Lorsque l'utilisateur aura frappé, par exemple,...

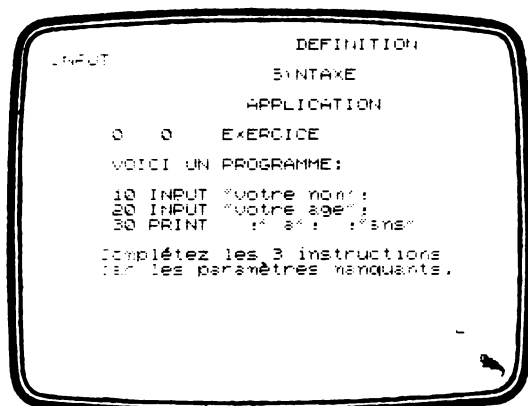
7 ENTREE

... la machine interprétera l'instruction qui lui commande d'afficher le mot COUT : suivi du produit de P par Q c'est-à-dire 84 (ce qu'elle fera !) :

COUT : 84

OK

EXERCICE



Les paramètres manquants sont les noms de variables dont la valeur est entrée dans les instructions 10 et 20. Ces noms doivent être répétés, dans l'instruction d'affichage 30, dans l'ordre adéquat pour produire une phrase du type "Jean a 17 ans".

Ici on pouvait frapper, par exemple :

```
N $ . A . N $ . A ENTREE
```

Attention à ne pas oublier le \$ à la fin du premier nom de variable, puisqu'elle est de type alphabétique !

INKEY\$

INKEY\$ est un nom de variable alphanumérique ; mais c'est une variable spéciale puisqu'elle ne prend pas sa valeur comme les autres variables (c'est-à-dire par une affectation de valeur avec =, ou une entrée INPUT) : elle est égale à la valeur de la touche frappée sur le clavier.

Évidemment elle a la plupart du temps la valeur 0 ; et si on veut retenir en mémoire un caractère frappé au clavier il faut mettre sa valeur dans une autre variable pendant que la touche est enfoncée.

Cela explique pourquoi INKEY\$ est souvent employé dans une instruction qui ressemble à celle qui est donnée en application :

```
10 A$ = INKEY$ : IF A$ = "" THEN 10
```

La première partie de l'instruction assure la mise en mémoire du caractère de la touche enfoncée ; la deuxième partie (qui emploie une instruction détaillée dans le chapitre suivant) assure l'exécution de l'instruction 10 tant qu'aucune touche enfoncée n'est repérée.

En français : Si A\$ est vide (c'est-à-dire si aucune touche n'est actuellement enfoncée) alors exécuter encore l'instruction numéro 10.

DATA READ

“DATA” signifie “données”.

“READ” signifie “lire”.

Ces deux instructions sont inséparables.

Écrire une liste de données (DATA) sans la lire (READ) n'est pas très intelligent.

Essayer de lire (READ) des données qui n'existent pas (DATA) provoque un message d'erreur.

Pour mettre en mémoire une liste de noms (par exemple vos amis), vous pourriez utiliser une série d'instructions d'affectation :

10 A\$ = “PIERRE”

11 B\$ = “ALAIN”

12 C\$ = “RICHARD”

13 D\$ = “MARC”

Cependant il est plus commode (plus clair et plus souple) de faire une véritable liste et de lire cette liste dans l'ordre de ses éléments.

Il suffit de prendre garde à ce que les éléments de la liste correspondent bien (un par un) aux noms qui leur sont donnés.

Ainsi les quatre instructions précédentes sont équivalentes aux deux suivantes :

10 DATA “PIERRE”, “ALAIN”, “RICHARD”, “MARC”

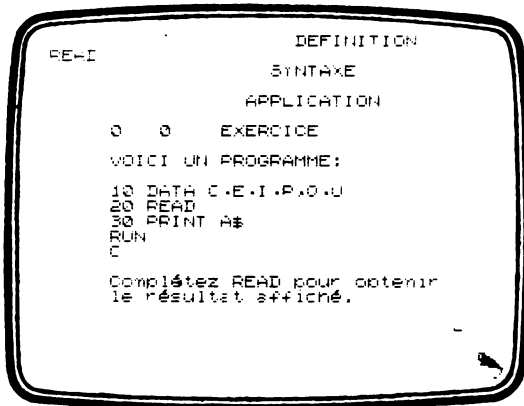
11 READ A\$, B\$, C\$, D\$

A l'exécution, la machine agira ainsi :

Les instructions DATA sont ignorées.

Chaque fois qu'elle interprète un nom de variable dans une instruction READ, elle lui donne pour valeur la donnée suivant la dernière donnée qu'elle a lue dans une instruction DATA.

EXERCICE



Il s'agit pour vous de repérer le nom de la variable affiché par l'instruction 30 et de l'indiquer à sa place dans l'instruction READ.

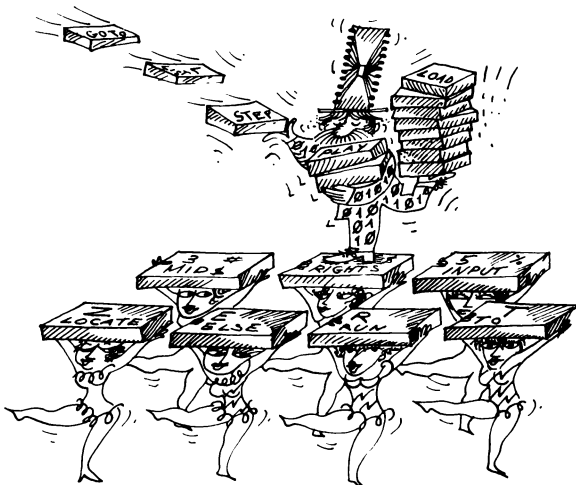
Ici il vous suffit de frapper

A \$ ENTREE

Mais, si un I avait été affiché après RUN, il aurait fallu frapper par exemple :

K \$, L \$, A \$ ENTREE

Attention à bien faire correspondre les noms de variables numériques aux données numériques et les noms de variables alphanumériques (avec un \$) aux données alphabétiques.



RESTORE

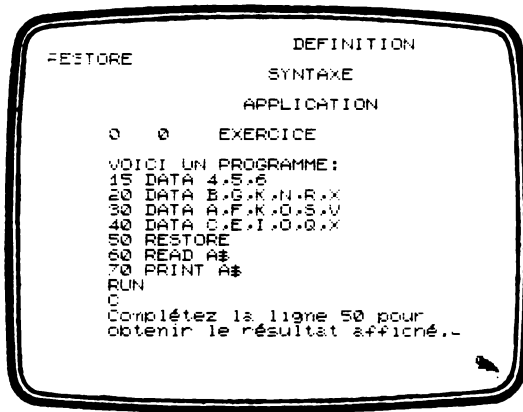
“RESTORE” signifie “remettre à neuf”.

Normalement la rencontre d’un nom de variable dans un READ provoque la lecture “de la donnée suivante” dans la liste des DATA.

La machine possède donc un “pointeur” (c’est-à-dire une sorte de marque) qui “pointe” sur le début de la prochaine donnée à lire et qui se déplace d’une donnée chaque fois qu’une variable prend sa valeur dans un READ.

L’instruction RESTORE permet de “casser” l’automatisme de déplacement du pointeur en commandant de le déplacer à un endroit précis (au début du programme ou au début d’une instruction de numéro précisée).

EXERCICE



```
RESTORE          DEFINITION
                  SYNTAXE
                  APPLICATION
0  0  EXERCICE
VOICI UN PROGRAMME:
15 DATA 4.5.6
20 DATA B.G.K.N.R.X
30 DATA A.F.K.O.S.V
40 DATA C.E.I.O.Q.X
50 RESTORE
60 READ A$
70 PRINT A$
RUN
C
Complétez la ligne 50 pour
obtenir le résultat affiché..
```

L’instruction 50 positionne le pointeur de données.

C’est à vous de découvrir à quel numéro d’instruction doit renvoyer ce pointeur en fonction de ce qui est lu en 60 et de ce qu’il est commandé d’afficher en 70.

Ici le “C” affiché fait partie de la liste commençant en 40 et on a lu une seule donnée (la première). Il vous faudrait donc frapper :

4 0 ENTREE .



INPUT or not INPUT ?

Pour bien saisir les avantages et les inconvénients de l'utilisation de l'instruction INPUT, nous allons reprendre un programme tout à fait simple mais très caractéristique.

La situation est la suivante : vous achetez des choses valant P francs l'unité et vous en achetez Q unités.

Le prix à payer vous est donné par le "programme" suivant :

```
1Ø INPUT P,Q  
2Ø PRINT P*Q
```

Pour connaître le prix de 3 kg de pommes à 6,40 francs le kilo, vous frapperez :

```
RUN ENTREE  
6.4,3 ENTREE
```

Pour connaître le prix de 35 boîtes à 2,80 francs la boîte, vous frapperez :

```
RUN ENTREE  
2.8,35 ENTREE
```

Maintenant mettez-vous à la place du vendeur de pommes ; à chaque client qui se présente (ou le soir lorsqu'il fait ses comptes) il doit frapper :

le prix, la quantité **ENTREE**

Comme, pour lui, le prix est toujours le même, il trouve fastidieux de le frapper à chaque exécution.

Il vaudrait mieux que le prix soit fixé dans le programme et non pas entré à chaque utilisation. Le programme du marchand de pommes est alors le suivant :

```
5 P = 6.4  
1Ø INPUT Q  
2Ø PRINT P*Q
```

Évidemment le prix change de temps en temps ; le vendeur de pommes devra alors changer l'instruction numéro 5 de son programme.

Mais il préfère changer une instruction après de nombreuses exécutions plutôt que de répéter la frappe du prix à chaque exécution.

Inversement, il serait fastidieux de changer une instruction pour changer la quantité à chaque exécution.

C'est donc au programmeur de bien savoir comment, pourquoi et par qui sera utilisé le programme pour bien doser les variables qui reçoivent leurs valeurs par le programme et celles qui les reçoivent par arrêt de l'exécution.

Les DATA

Les conclusions du paragraphe précédent sont évidemment valables pour le choix de données entrées par INPUT ou fixées par des DATA.

Par exemple, vous voulez calculer le montant de vos impôts (et celui de vos amis). Le "bon" programme de calcul aura la structure suivante :

10 DATA (les tranches d'imposition)

20 INPUT (les éléments de votre revenu)

100 ' CALCUL DE L'IMPOT

200 ' Affichage des résultats

... et tous les ans vous changerez la ligne 10 en réécrivant les limites des nouvelles tranches d'imposition.

Il serait donc très maladroit d'obliger l'utilisateur à rentrer en INPUT les limites des tranches.

(Seul un fonctionnaire du ministère des finances pourrait trouver un intérêt à une instruction 10 INPUT..., si justement il voulait connaître les effets d'un changement des tranches sur le montant de l'impôt.)

Le bouclage sur les entrées

Lorsqu'un programme doit être utilisé plusieurs fois de suite avec des données différentes, on a intérêt à "boucler" sur les entrées.

On économise ainsi la frappe de RUN **ENTREE** qui finit par être lassante.

Ainsi le programme du consommateur devient :

```
10 INPUT P,Q  
20 PRINT P*Q  
30 GOTO 10
```

Pour connaître les prix de différents achats, on lance l'exécution "une fois pour toutes". Par exemple :

```
RUN ENTREE  
6.4,3 ENTREE  
affichage : 19.2  
2.8,35 ENTREE  
affichage : 98  
7,2.5 ENTREE  
affichage : 17.5
```

EXPÉRIENCE

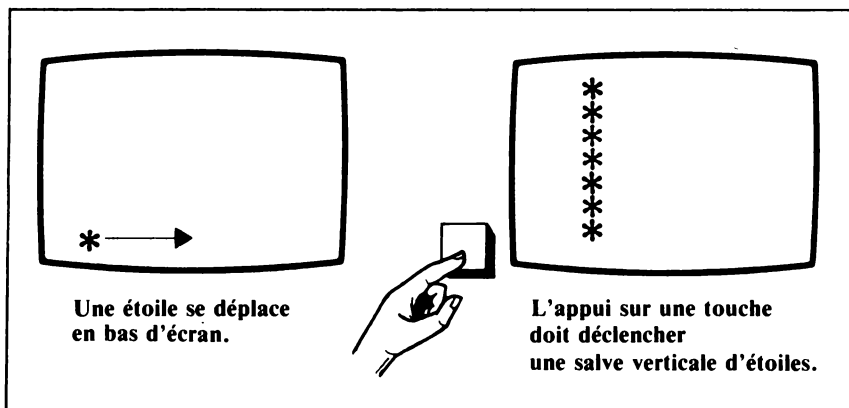
L'exécution du programme suivant devrait vous faire réfléchir sur les considérations du premier paragraphe :

```
5 INPUT P  
10 INPUT Q  
20 PRINT P*Q  
30 GOTO 10
```


La guerre des étoiles

L'instruction INKEY\$ est utile pour déclencher certaines actions lorsque l'utilisateur appuie sur la touche. Par exemple pour envoyer un (soi-disant) projectile au moment opportun.

Traisons ici le cas le plus simple mais aussi le plus général, celui qu'il suffira d'imiter en le sophistiquant d'effets divers.



EXPÉRIENCE

10 P = 0

20 PSET (P,24) "*"

30 P = P + 1

40 A\$ = INKEY\$: IF A\$ < > "" THEN LINE (P,24)-(P,0) "*"

50 GOTO 20

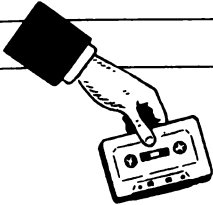


Remarque : le signe < > signifie "différent de".

L'instruction 40 commande donc de tracer une ligne verticale d'étoiles si A\$ n'est pas vide, c'est-à-dire si on a appuyé sur une touche.

CHAPITRE 9

Branchements



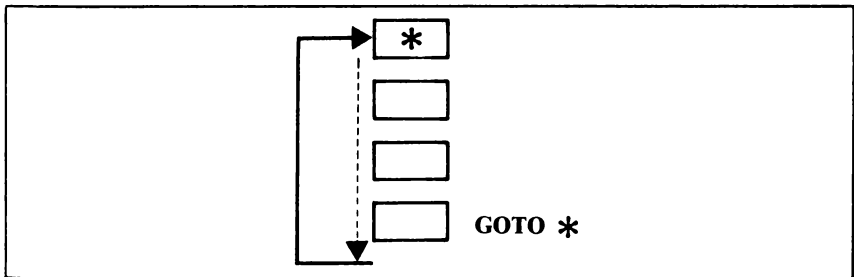
Suivi de la cassette

Les “branchements” permettent de faire (ou de refaire) telle ou telle chose, selon qu’une certaine condition est vérifiée ou non (ou plus généralement, selon la valeur d’une certaine variable).

GOTO

“GOTO” signifie “aller à”.

Nous avons, en fait, déjà utilisé cette instruction dans les chapitres précédents (si nous ne l’avions pas fait, nos programmes auraient été bien tristes !).



Cette instruction permet donc de répéter indéfiniment une “boucle” d’instructions.

L’ennui de cette instruction, c’est que rien de naturel ne l’arrête. En fait, l’arrêt se produira... ou bien volontairement en appuyant sur CNT-C... ou bien involontairement parce qu’une chose impossible à faire sera demandée (calculs sur de trop grands nombres, dessins hors de l’écran...) et un message d’erreur sera affiché.

C'est pourquoi lorsqu'on écrit de "vrais" bons programmes, on n'utilise jamais l'instruction GOTO !

IF... THEN... ELSE...

"IF... THEN... ELSE..." signifie "si... alors... sinon...".

Attention : il y a trois pages de syntaxe pour cette instruction ! et c'est la 3^{ème} page la plus intéressante.

La ligne :

```
IF A <> B THEN 10 ELSE PRINT A : GOTO 30
```

signifie :

"Si A n'est pas égal à B, alors aller à 10, sinon afficher la valeur de A, puis aller à 30."

Il est possible d'écrire plusieurs instructions (séparées par deux points) après THEN ou après ELSE. Par exemple :

```
1 INPUT NATE
2 IF NATE < 10 THEN PRINT "COLLE" : GOTO 1 ELSE
  PRINT "RECU" : GOTO 1
```

Ou bien on peut "imbriquer" plusieurs conditions les unes dans les autres :

```
2 IF NATE < 10 THEN PRINT "COLLE" : GOTO 1 ELSE
  PRINT "RECU" : IF NATE > = 16 THEN PRINT " TRES
  BIEN" : GOTO 1
```

On peut aussi se faire succéder les conditions :

```
2 IF NATE < 10 THEN PRINT "COLLE" : GOTO 1 ELSE
  PRINT "RECU"
3 IF NATE > = 16 THEN PRINT " TRES BIEN" : GOTO 6
4 IF NATE > = 14 THEN PRINT " BIEN" : GOTO 6
5 IF NATE > = 12 THEN PRINT " ASSEZ BIEN"
6 GOTO 1
```

Attention

Pour tenir compte d'une condition vous avez toujours deux choix. En trois exemples vous pouvez vous interroger sur la vérité de "NATE > = 16" ou de "NATE < 16".

Contrairement à ce que l'on pourrait croire, les deux solutions ne sont pas équivalentes. En effet, la possibilité de sous-entendre le ELSE (mais jamais le THEN) permet parfois de bien simplifier les écritures.



Remarque

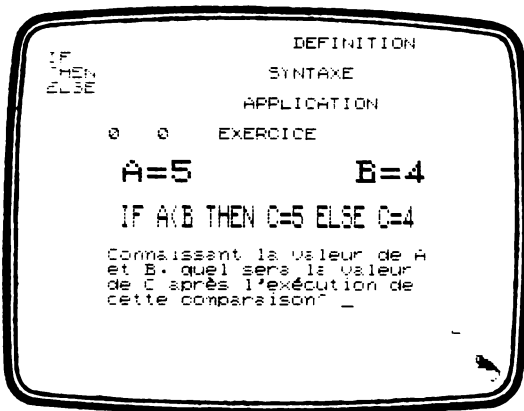
En regardant sur votre manuel, vous risquez de découvrir pourquoi nous avons noté la NOTE, NATE. (Oui ! cette note est un gag.)

On peut enfin utiliser les opérations logiques (AND, OR, NOT) pour "affiner" les conditions :

```
3 IF NATE < 16 AND NATE > = 14 THEN PRINT " BIEN"
```

(voir le chapitre 4).

EXERCICE



Ici, A n'étant pas plus petit que B, C prend la valeur 4. Il faut donc frapper :

4 ENTREE

ON... GOTO...

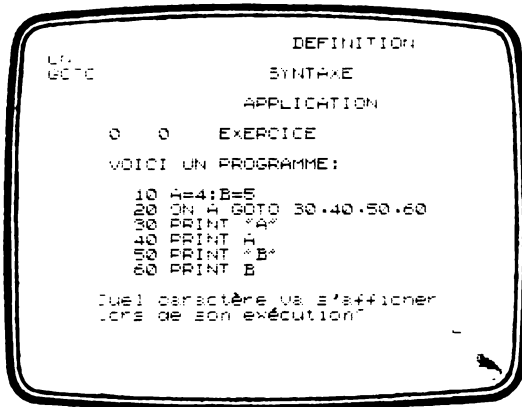
“ON TRUC GOTO...” signifie, en anglais : “selon truc aller à...”

Cette instruction permet de débrancher l'exécution vers des actions différentes selon la valeur de la variable nommée entre ON et GOTO.

Le programme suivant montre les possibilités ouvertes :

```
1 INPUT NATE
2 IF NATE < 10 THEN PRINT "COLLE" : GOTO 1
  ELSE PRINT "RECU"
3 TRUC = NATE - 9
4 ON TRUC GOTO 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20
10 PRINT "    MAIS C'EST VRAIMENT TRES JUSTE !" :
  GOTO 1
11 PRINT "    MAIS VOUS AURIEZ PU FAIRE BEAUCOUP
  MIEUX" : GOTO 1
12 PRINT "    MENTION AB, C'EST PAS MAL, MAIS IL Y A
  ENCORE MIEUX" : GOTO 1
13 ...
(Nous vous laissons le soin d'écrire les instructions 13 à 20.)
```

EXERCICE



La valeur de A, prise dans l'instruction 10 vous permet de trouver vers quelle instruction le branchement va être commandé en 20 et vous en concluez le premier caractère affiché (voir les compléments : le piège à conditions).

Ici, A vaut 4, l'instruction 20 branche donc au quatrième numéro, c'est-à-dire 40 et c'est donc la valeur de A qui va s'écrire.

Il faudrait frapper :

4 ENTREE .

(Attention : PRINT "A" affiche la lettre A
PRINT A affiche la valeur de A.)



Le piège à conditions

EXPÉRIENCE

```
10 INPUT TEST
20 IF TEST = 1 THEN 30 ELSE 40
30 PRINT "UN"
40 PRINT "BOF"
50 END
```

Avant de faire exécuter ce programme, regardez-le bien et essayez de répondre aux questions suivantes :

a. Si on frappe...

R U N ENTREE

? 2 ENTREE

... que va afficher la machine ?

b. Si on frappe...

R U N ENTREE

? 1 ENTREE

... que va afficher la machine ?

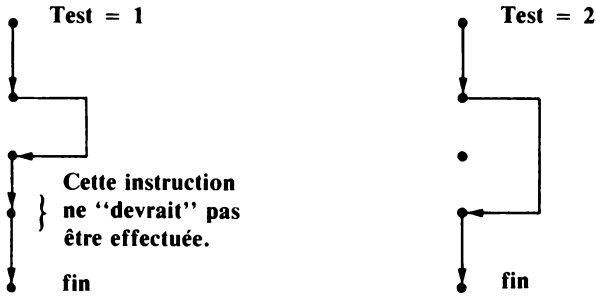
Si vous avez répondu "BOF" à la question a et "UN" à la question b, alors vous avez... perdu ! En effet, vous êtes tombé dans le "piège à conditions". Voici ce qui se passe lorsque TEST prend la valeur 1 :

Dans l'instruction 20, la condition TEST = 1 est vérifiée et donc le branchement se fait en 30.

En 30, on affiche bien "UN". Mais ensuite ?

La machine exécute alors l'instruction suivante, c'est-à-dire l'instruction 40 et elle affiche aussi "BOF" avant de s'arrêter.

Les cheminements selon la valeur de TEST :



Ce piège vous est tendu par les dessous de la programmation dès que vous utilisez une instruction IF... THEN... ELSE... (ou bien ON... GOTO..., comme dans l'exercice donné dans ce chapitre).



Attention

Suivez bien les deux "branches" suivant une instruction conditionnelle (jusqu'à leur regroupement ou leur extrémité) pour que l'une ne marche pas sur les pieds de l'autre.

Ici par exemple, on aurait pu écrire :
30 PRINT "UN" : GOTO 50

Mais on aurait aussi pu allonger l'instruction 20 de manière à ne pas avoir à brancher l'exécution à d'autres numéros de lignes :
20 IF TEST = 1 THEN PRINT "UN" ELSE PRINT "BOF"

Le conseil est donc le suivant : chaque fois que cela paraît raisonnable, faites suivre THEN et ELSE de l'énoncé de l'ensemble des actions à accomplir selon la valeur de la condition.

La boucle ...TANT QUE...

Les répétitions commandées jusqu'à ce chapitre par l'instruction GOTO... n'avaient pas de fin.

Or il arrive souvent que l'on veuille accomplir certains traitements tant qu'une certaine condition est réalisée et, donc, arrêter ces traitements dès qu'elle ne l'est plus.

L'instruction IF... THEN... ELSE... permet de programmer de telles boucles.

En français :	En BASIC (version 1).
1	1
2	2
3 TANT QUE (condition) REPETER :	3 IF (condition) THEN 4 ELSE 8
[4 5 6 7]	4 5 6 7 .. : GOTO 3
8	8
9	9
10	10
.	.
.	.

Si on est sûr que la boucle 4 5 6 7 doit être exécutée au moins une fois, il vaut mieux tester la vérité de la condition en fin de boucle, comme ceci :

En BASIC (version 2)	
1	
2	
3	
4	
5	
6	
7	... : IF (condition) THEN 4
8	
9	
10	
.	
.	

Les boucles de ce type interviennent fréquemment dans les programmes. On peut vouloir en effet répéter quelque chose...

... tant que tout va bien !

... tant qu'un certain compteur n'a pas atteint une certaine valeur,

... tant qu'on n'est pas suffisamment près d'un objectif à atteindre,

... tant qu'une intervention extérieure ne se manifeste pas.

Voici trois exemples simples de ces possibilités.

Exemple : compter de tant à tant

L'affichage d'une table de racine carrée (programme d'application de l'instruction GOTO) depuis 0 jusqu'à 100 s'écrit ainsi :

```
10 I = 0
```

```
20 I = I + 1
```

```
30 PRINT I, SQR (I)
```

```
40 IF I < 100 THEN 20
```

Autre exemple : tendance vers une limite

```
10 INPUT A, B, C, Z, : K = 0
```

```
20 BOXF (A - C/2, B - C/2) - (A + C/2, B + C/2), K
```

```
30 C = Z*C : K = (K + 1) MOD 16
```

```
40 IF C > 7 THEN 20
```

```
50 GOTO 10
```

Essayez ce programme avec :

```
RUN ENTREE
```

```
150, 100, 90, .8 ENTREE
```

```
100, 120, 60, .5 ENTREE
```

... (et analyser chacune des instructions).

Autre exemple : Interruption au clavier

```
10 COLOR 1,2 : LOCATE 0,0 : PRINT BOF
```

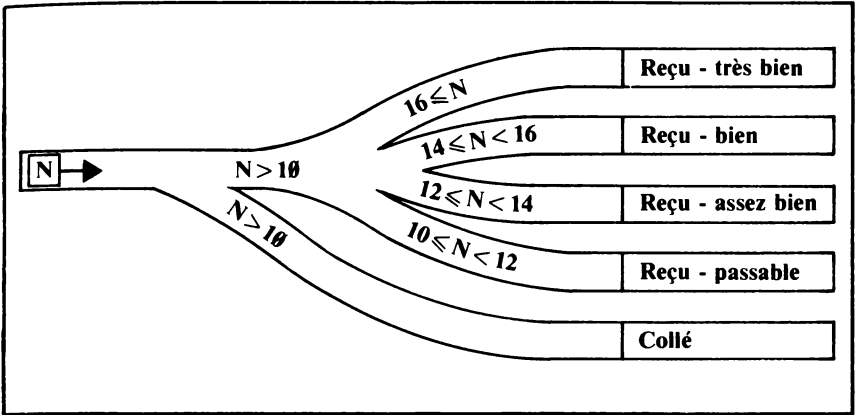
```
20 COLOR 2,1 : LOCATE 0,0 : PRINT BOF
```

```
30 BOF = BOF + 1
```

```
40 A$ = INKEY$
```

```
50 IF A$ = "" THEN 10 ELSE STOP
```

Aiguillages



Le dessin ci-dessus et le programme ci-dessous valent la peine d’être examinés et étudiés. Vous ne le regretterez sûrement pas.

```
1 INPUT N
2 ON N @ 10 + 1 GOTO 3,4
3 ? "COLLE" : GOTO 1
4 ? "RECU"
5 ON (N - 10) @ 2 + 1 GOTO 6, 7, 8, 9
6 ? " P" : GOTO 1
7 ? " AB" : GOTO 1
8 ? " B" : GOTO 1
9 ? " TB" : GOTO 1
```



Remarque

Comme dans les instructions 2 ou 5, vous aurez souvent à ajouter 1 ou à enlever 1 pour “ajuster” vos formules. Le mieux est de faire un essai pour contrôler ou corriger votre “tir”.

Ici, par exemple, vous pensez :

“Les choses marchant 2 par 2 au dessus de 10 il faut sûrement que je divise $N - 10$ par 2.

Si N vaut 15, $N - 10$ vaut 5, $(N - 10) @ 2$ vaut 2 et je suis dans le 3^{ème} cas (B après P et AB).

Il faut donc ajouter 1”.

Plus, moins ou égal ?

Étant donné deux nombres, il arrive parfois que l'on veuille traiter séparément les trois cas possibles :

- l'un est plus grand que l'autre,
- ils sont égaux,
- l'autre est plus grand que l'un.

La distinction $A < B$, $A = B$, $A > B$ peut être faite par deux IF... successifs mais il est sûrement plus élégant d'utiliser un ON TRUC GOTO... en calculant TRUC de manière astucieuse.

Pour cela, il faut se souvenir que, dans le MO5 et le TO7(-70), la valeur logique "VRAI" vaut en fait la valeur numérique -1 et que "FAUX" vaut 0.

Voilà le truc :

$$10 \text{ TRUC} = -(A < B) - 2*(A = B) - 3*(A > B)$$

20 ON TRUC GOTO 100, 200, 300

100 ' cas où A est plus petit que B

.....

200 ' cas où A est égal à B

.....

300 ' cas où A est plus grand que B

.....



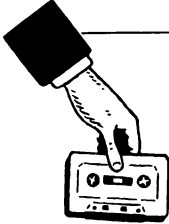
Remarque

Afin d'éviter le "piège à conditions", il vaut mieux utiliser l'instruction ON TRUC GOSUB... (qui sera vue au chapitre suivant) plutôt que l'instruction ON TRUC GOTO...

CHAPITRE 10

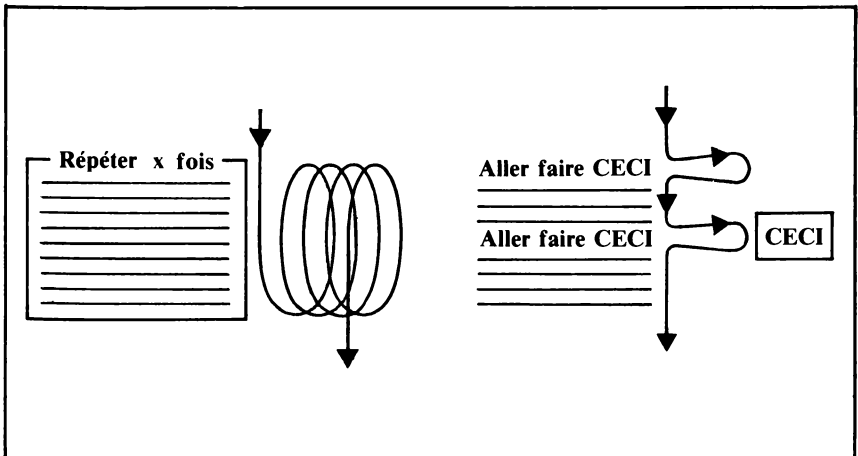
Instructions répétitives

Suivi de la cassette



Voici maintenant deux instructions très utiles pour l'écriture raisonnée de programmes, dès qu'ils commencent à se compliquer un peu.

La première de ces instructions concerne la notion de *boucle* : il s'agit d'exécuter plusieurs fois une même suite d'instructions pendant qu'une certaine variable (un "compteur") parcourt un certain ensemble de valeurs. La deuxième concerne la notion de *sous-programme* : il s'agit d'exécuter chaque fois qu'on le souhaite une même suite d'instructions.



FOR... NEXT

“FOR N=X TO Y STEP Z... NEXT N”

signifie :

“DE N=X À Y PAR PAS DE Z... N SUIVANT”

Jusqu'à maintenant, il était possible de faire exécuter une boucle d'instructions grâce à l'instruction IF... THEN... ELSE... Le cas particulier de boucles dont il est ici question est le suivant :

exemple	cas général
En français	
Faire [quelque chose] pour N=3, pour N=5, pour N=7, pour N=9, pour N=11 .	Faire [quelque chose] pour N=X, pour N=X+Z, pour N=X+2*Z,... pour N=Y.
En français encore, proche du BASIC	
Faire, de N=3 à 11 par pas de 2 [quelque chose]	Faire, de N=X à Y par pas de Z [quelque chose]
En BASIC	
FOR N=3 TO 11 STEP 2 quelque chose : NEXTN	FOR N=X TO Y STEP Z quelque chose : NEXTN

Remarque

l'instruction NEXT N indique la fin de la boucle d'instructions (comme une parenthèse ou un crochet fermant) : Si N n'a pas atteint sa valeur maximale, on passe au N suivant et on revient en début de boucle ; sinon on quitte la boucle et on exécute la suite du programme.

Attention

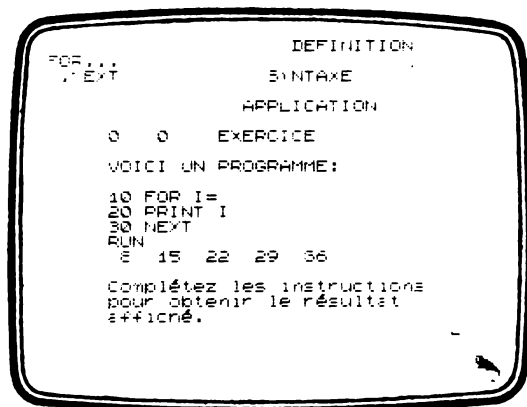
Il y a deux pages "application" dans le programme d'auto-initiation ; l'une est numérique, l'autre est graphique.

Remarquez, dans le programme d'application faisant clignoter la tête du caméléon, les instructions 40 et 70 :

```
FOR J=1 TO 600 : NEXT J
```

Il s'agit d'une boucle de temporisation ; on fait bêtement compter la machine jusqu'à 600 (comme celui "qui s'y colle", à cache-cache) pour laisser le temps à l'utilisateur de contempler l'écran.

EXERCICE



```
DEFINITION
FOR...
NEXT
SYNTAXE
APPLICATION
0 0 EXERCICE
VOICI UN PROGRAMME:
10 FOR I=
20 PRINT I
30 NEXT
RUN
E 15 22 29 36
Complétez les instructions
pour obtenir le résultat
affiché.
```

C'est à vous de compléter la ligne 10 compte tenu de l'affichage obtenu après RUN.

Ici I prend les valeurs de 8 à 36 par pas de 7. Il faudrait donc frapper :

8 TO 36 STEP 7 **ENTREE**.

GOSUB... RETURN

En anglais, “GOSUB...” signifie “aller sous...” et “RETURN” signifie “retour”. “Aller sous...” est l’abréviation de “aller exécuter le sous-programme”.

La signification du petit mot “sous” peut être ici rapprochée de sa signification dans l’expression “sous-traitant”.

Le programme principal agit en effet comme une “entreprise”, effectuant successivement tous les travaux qui lui sont demandés.

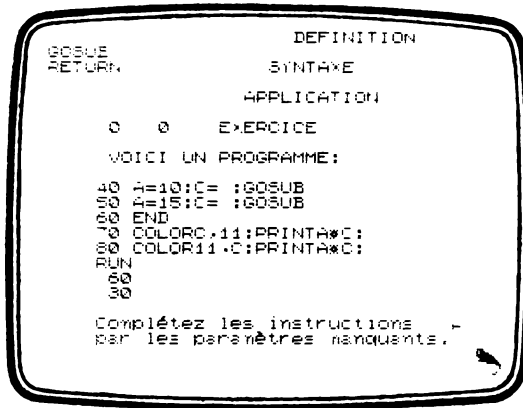
<i>Programme principal</i>	
1	instruction
2	instruction
3	instruction
⋮	
14	instruction
15	GOSUB 100
16	instruction

Ici, après avoir exécuté les 14 premières instructions, le programme principal rencontre l’instruction numéro 15. Cette instruction lui ordonne de “sous-traiter” le travail à effectuer au “sous-programme” qui commence à l’instruction 100. C’est donc ce qu’il fait... et il attend tranquillement le signal de fin du travail sous-traité avant de se remettre au travail principal avec l’exécution de la ligne 16...

<i>sous-programme</i>	
100	instruction
⋮	
199	RETURN

Le signal de fin du travail sous-traité sera émis par le sous-traitant lorsqu’il rencontrera l’instruction RETURN qui conclut donc obligatoirement le sous-programme.

EXERCICE



Vous avez à compléter les instructions 40 et 50 après avoir très fortement réfléchi à l’affichage obtenu qui est, dans chacun des deux cas, le produit $A * C$.

Ici, le premier produit affiché vaut 60 alors que A vaut 10, donc C vaut 6.

D’autre part la couleur numéro 11 est la couleur 3 clair c’est-à-dire jaune clair (et “6” est le numéro de la couleur “cyan”).

Si, par exemple, “60” est écrit en cyan sur jaune clair, c’est que l’instruction 70 a été d’abord exécutée (si “60” avait été écrit en jaune clair sur cyan, c’est l’instruction 80 qui aurait été d’abord exécutée).

La ligne 40 devrait donc être complétée ainsi :

6 → 7 0 →

Et, si “30” est écrit en jaune clair sur jaune foncé, la ligne 50 devrait être complétée ainsi :

2 → 8 0 ENTREE



Ifoufor ?

La séquence d'instructions...

```
30 FOR N = X TO Y STEP Z
```

```
31 ...
```

```
...
```

```
39 NEXT Z
```

... est complètement équivalente à la séquence...

```
30 N = X
```

```
31 ...
```

```
...
```

```
39 N = N + Z : IF N <= Y THEN 31
```

On pourrait alors se demander pourquoi les instructions de type FOR... NEXT ont été créées.

Mais il faut bien reconnaître que la séquence avec FOR... NEXT est plus proche de notre façon de penser ; il s'agit donc essentiellement d'une facilité mentale pour l'écriture de programmes qui, de plus, les rend plus simples à interpréter à la lecture.

Une petite différence existe toutefois entre les deux séquences :

Si une instruction interne à la boucle (par exemple, ici, de numéro 35) renvoie l'exécution à l'extérieur de la boucle (par exemple au numéro 60), alors il vaut mieux utiliser IF... THEN... En effet, en utilisant FOR... NEXT on encombre la mémoire puisqu'il faut conserver la valeur maximale Y qui ne sera (peut-être) jamais atteinte.

Quelques cas "limites"

Quelle est la valeur de la "variable de boucle" à la sortie de la boucle ?

La boucle est-elle parcourue si les bornes minimales et maximales sont égales ?

Et si la borne maximale est plus petite que la borne minimale ?

Et si le pas est négatif ?

Peut-on modifier la valeur de la variable de boucle à l'intérieur de la boucle ?

A toutes ces questions, les expériences suivantes apportent une réponse (si vous voulez bien y réfléchir !).

EXPÉRIENCES

10 N=0 : X=1 : Y=11 : Z=2

20 FOR N=X TO Y STEP Z : PRINT N : NEXT N

30 PRINT "SORTIE" ; N

40 END

Dans 40 :

a) remplacer Y=11 par Y=1

b) remplacer Y=1 par Y=0

c) remplacer X=1 par X=10

et Z=2 par Z=-2

d) remplacer Y=0 par Y=16

(attention : CNT-C)

e) remplacer Z=-2 par Z=4

Dans 20, insérer N=N+1 avant PRINT N.

Quelques "effets" graphiques

L'utilisation de boucles FOR... NEXT associées au changement de couleur et de position d'une figure permet d'obtenir de jolis effets.

EXPÉRIENCE

10 C=0

20 FOR X=0 TO 312 STEP 8

30 BOXF (X,0)-(X+7,199),C

40 C=(C+1) MOD 16

50 NEXT X

ACTIVITÉ

Écrire un programme analogue qui dessine des bandes horizontales.

EXPÉRIENCE

a) Remplacer 3Ø par :

3Ø BOXF (X, 5*X/8) – (319 – X, 199 – 5*X/8), C

b) Remplacer 3Ø par :

3Ø LINE (X, Ø) – (319, 5*X/8) : LINE – (319 – X, 199) :
LINE – (Ø, 199 – 5*X/8) : LINE – (X, Ø)

c) Remplacer 3Ø par :

3Ø LINE (X, Ø) – (319 – X, 199), C

Le travail en équipe

Il est vrai que l'instruction GOSUB vous permet de faire exécuter plusieurs fois un même travail à partir de points différents d'un programme.

Mais ce n'est pas l'intérêt principal de cette instruction, car celui-ci réside plutôt dans la souplesse et la qualité d'organisation des programmes que permet ce GOSUB.

Le paragraphe suivant montre l'exemple d'une bonne "structuration" du travail.

Dans ce paragraphe nous insistons sur les possibilités de "partage" du travail entre plusieurs programmeurs.

Imaginons, par exemple, que vous vouliez réaliser un certain "moiré" sur l'écran.

Pour cela, vous vous proposez de placer quelques points sur l'écran et de faire rayonner, à partir de chacun de ces points, un "faisceau" de droites allant jusqu'au bord de l'écran comme de multiples rayons.

Le partage du travail pourrait alors être le suivant :

“Moi, je fais le programme principal...

... et toi, tu me fais un sous-programme, écrit à partir de 10100 et qui dessine un faisceau de droites centré sur le point de coordonnées (X,Y).”

La “sous-traitance” étant lancée, je peux me contenter d’écrire le programme principal.

Le voici :

```
5 SCREEN 1,0,0
10 X = 160 : Y = 60
11 GOSUB 10100 'FAISCEAU
20 X = 100 : Y = 140
21 GOSUB 10100 'FAISCEAU
30 X = 200 : Y = 140
31 GOSUB 10100 'FAISCEAU
99 END
```

Et pendant que j’ai écrit le programme principal, tu as pu écrire le sous-programme “FAISCEAU” que voilà :

```
10100 'FAISCEAU
10110 FOR A = 0 TO 319 STEP 8 : LINE (X,Y) - (A,0) : NEXT A
10120 FOR A = 0 TO 199 STEP 8 : LINE (X,Y) - (319,A) : NEXT A
10130 FOR A = 319 TO 0 STEP - 8 : LINE (X,Y) - (A,199) : NEXT
  A
10140 FOR A = 199 TO 0 STEP - 8 : LINE (X,Y) - (0,A) : NEXT A
10199 RETURN
```



Remarque

Comme vous le constatez une même “variable de boucle” peut être utilisée dans des boucles indépendantes.

La pratique de la sous-traitance de morceau de programme exige évidemment de bien préciser les éventuelles variables d’entrées et de sorties des sous-programmes : ici, par exemple, les coordonnées du point central du faisceau s’appellent obligatoirement X et Y.

Petit cadeau pour réfléchir

Le programme principal pourrait devenir plus simple tout en offrant plus de possibilités.

```
5 SCREEN 1,0,0
10 DATA 2, 150, 100, 160, 100, 50, 50, 250, 50, 50, 150, 250, 150
11 READ N
20 FOR I=1 TO N
30 READ X,Y : GOSUB 10100
40 NEXT I
99 END
```

ACTIVITÉS

Changer le premier 2 des DATA en 4, en 5 ou en 6...

Provoquer des changements de couleurs pour chaque faisceau...

L'écriture descendante

Le paragraphe précédent illustre (en partie) une technique d'écriture des programmes dite "descendante". Il s'agit d'écrire d'abord un programme principal sans détailler les traitements qui nécessitent réflexion. On obtient ainsi un programme qui renvoie à des sous-programmes, lesquels seront écrits plus tard.

Ces sous-programmes peuvent d'ailleurs eux-mêmes faire appel à d'autres sous-programmes encore plus détaillés. On "descend" ainsi petit à petit vers le détail en n'ayant jamais à considérer plus d'un ensemble significatif d'instructions à la fois.

Cette méthode de programmation oblige évidemment à bien préciser la manière dont se suivent ou s'appellent les sous-ensembles d'instructions ; en particulier, il faut bien faire la liste des variables qui vont "passer" d'un ensemble à un autre.

Cela peut paraître contraignant au débutant mais, justement, ce sont de bonnes habitudes à prendre si vous voulez un jour vous lancer dans l'écriture de programmes importants !

Exemple

```
10 'Résolution du système d'équations à deux inconnues
20 'AX + BY = C
30 'DX + EY = F
40 INPUT A,B,C,D,E,F
50 DET = A*E - B*D
60 IF DET <> 0 THEN GOSUB 100 : GOTO 91 'UNE SOLUTION
70 DET 1 = A*F - C*D
80 IF DET 1 <> 0 THEN GOSUB 200 : GOTO 91 'PAS DE SOLUTION
90 GOSUB 300 : GOTO 91 'INFINITE DE SOLUTIONS
91 PRINT : PRINT "POUR UN AUTRE SYSTEME, FRAPPEZ
  UNE TOUCHE"
92 A$ = INKEY$ : IF A$ = "" THEN 92
93 GOTO 40
99 END
```

```
100 'UNE !
110 PRINT : PRINT "IL Y A UNE SEULE SOLUTION AU
  SYSTEME !"
120 PRINT "X = " ; (C*E - B*D)/DET
130 PRINT "Y = " ; (A*F - C*D)/DET
199 RETURN
```

```
200 'PAS !
210 PRINT : PRINT "LE SYSTEME N'A PAS DE SOLUTION !"
299 RETURN
```

```
300 'INFINITE !
310 PRINT : PRINT "LE SYSTEME ADMET POUR SOLUTION
  TOUT COUPLE DE LA FORME :'"
320 PRINT "X = " ; C/A ; "+K.(“; B ;”)”
330 PRINT "Y = " ; “K.(“; - A ;”)”
340 PRINT "K ETANT UN NOMBRE QUELCONQUE !"
399 RETURN
```



Remarque

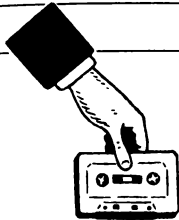
Pour bien marcher dans tous les cas, ce dernier sous-programme doit être complété par :

```
315 IF A = 0 THEN PRINT "X QUELCONQUE" : IF B = 0 THEN  
PRINT "Y QUELCONQUE" ELSE PRINT "Y =" ; C/B
```

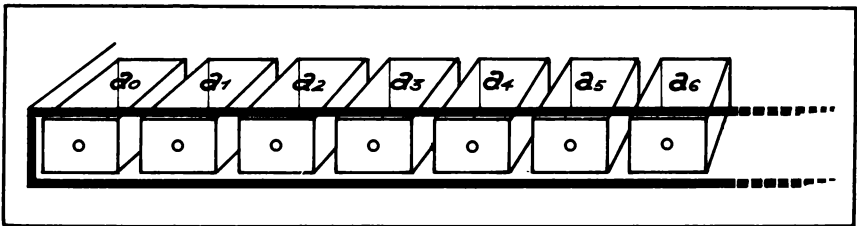
CHAPITRE 11

Tableaux

Suivi de la cassette



Nous allons maintenant ranger les variables dans des tiroirs numérotés.



Si l'on doit travailler sur les jours de la semaine, il est sûrement plus pratique de parler du "jour de la semaine numéro 3" que du mercredi ; surtout si l'on doit parler du jour suivant ou de 10 jours après...

Ce chapitre va nous permettre d'appeler :

- "LUNDI" sous le nom J\$ (1).
- "MARDI" sous le nom J\$ (2).
- "MERCREDI" sous le nom J\$ (3).
- "JEUDI" sous le nom J\$ (4).
- "VENDREDI" sous le nom J\$ (5).
- "SAMEDI" sous le nom J\$ (6).
- "DIMANCHE" sous le nom J\$ (7).

Ainsi, par exemple, le dixième jour après J\$ (1) est le jour $J\$ ((1 + 10) \text{ MOD } 7)$.



Remarque

Le début de programme suivant donne leur nom à chacun des jours.

```
10 DATA DIMANCHE, LUNDI, MARDI, MERCREDI, JEUDI,  
    VENDREDI, SAMEDI
```

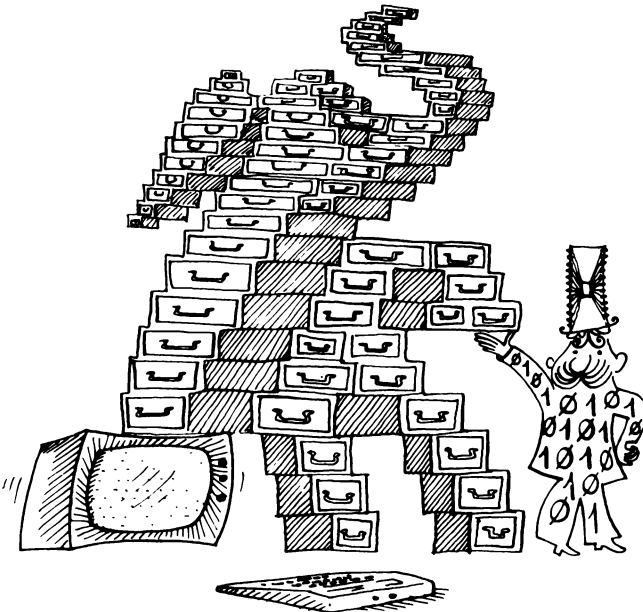
```
11 FOR I=0TO6 : READ J$( I) : NEXT I
```

Dans J\$(I), le nombre I est l'indice et J\$ est le nom du tableau des jours de la semaine.

DIM

“DIM” est l'abréviation de “DIMension”.

Normalement la machine attribue un “ tiroir ” (des mémoires) à chaque nouvelle variable apparaissant dans le programme.



Ainsi, si vous parlez de J\$ (3), elle range "mercredi" dans un tiroir, puis si vous parlez de B elle lui attribue un autre tiroir, puis si vous parlez de J\$ (5) un autre et ainsi de suite.

Mais, dans la suite du traitement, il y a évidemment intérêt à ce que les éléments d'un même tableau soient rangés les uns à côté des autres (ils sont retrouvés plus facilement !). C'est pourquoi il est prévu d'annoncer à la machine que l'on va utiliser un tableau afin qu'elle réserve la place pour ranger tous les éléments ; et il faut donc lui indiquer le nombre maximum d'éléments que pourra contenir le tableau.

L'instruction...

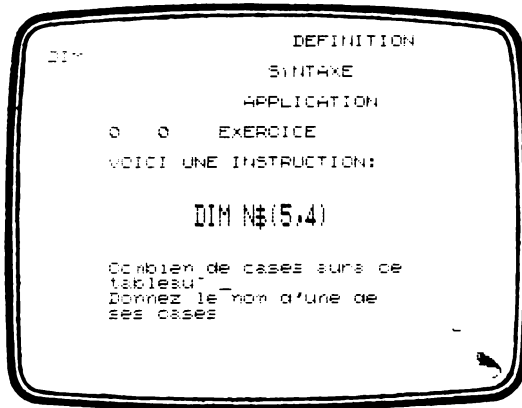
DIM B (17)

... réserve donc une suite de 18 tiroirs pour ranger chacune des variables numériques du tableau B.

Comme vous le voyez la numérotation des variables d'un tableau débute à 0.

Un bon conseil : Habituez-vous à penser que le premier élément d'une liste porte le numéro zéro.

EXERCICE



Ici, on crée un tableau alphanumérique ayant 6×5 éléments. Il faudrait donc frapper, par exemple :

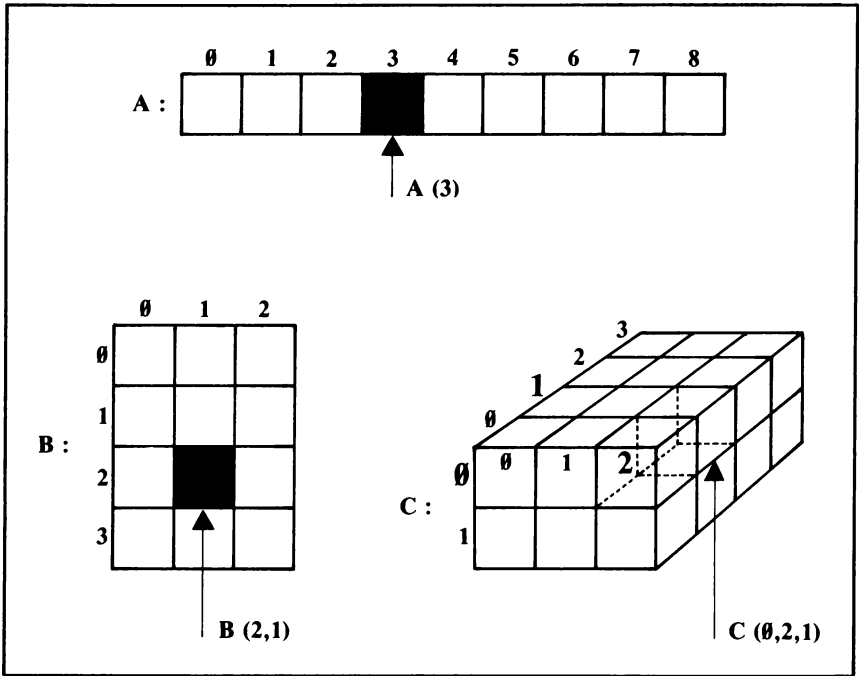
3 0 N \$ (1 , 2) ENTREE

(A la place de **1** on pourrait frapper tout chiffre entre 0 et 5, à la place de **2** tout chiffre entre 0 et 4).

A (i) A\$(j)

Les tableaux peuvent être à 1 indice, 2 indices, 3 indices...

Lorsque l'homme se les imagine, il pense en général à ceci :



Pour un nombre d'indice supérieur, "l'intuition" spatiale ne marche pas aussi bien. Mais il y a d'autres manières de penser. Par exemple, $L(6, 2937, 37, 23, 7)$ peut très bien être la 7^{ème} lettre de la ligne numéro 23 de la page 37 du 2937^{ème} livre de la bibliothèque française numéro 6 !



Remarque

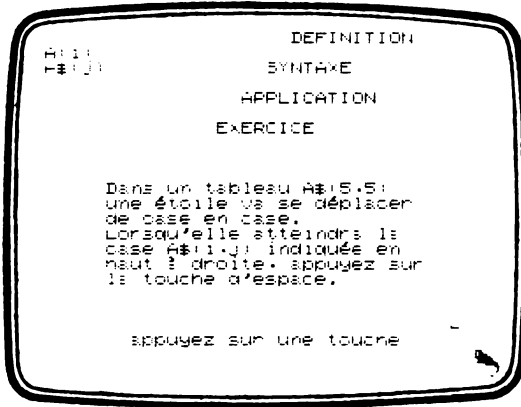
Dans le programme d'application, l'instruction $L = \text{INT}(\text{RND} * 5)$ choisit au hasard un numéro de ligne entre 0 et 4.



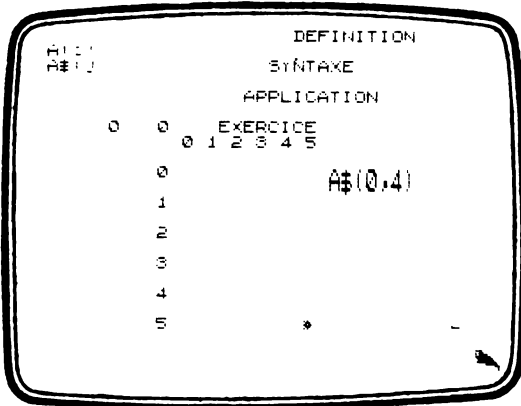
Attention

Dans ce chapitre, il y a trois pages successives d'applications !

EXERCICE



Après la première page, appuyez sur une touche.

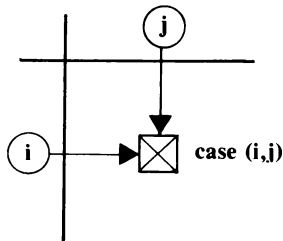


La case indiquée (0,4) est sur la ligne 0 et la colonne 4. Il faudra donc attendre que l'étoile, actuellement en (5,5), arrive dans cette case pour frapper la touche ████.

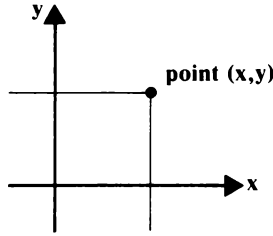


Remarque

L'habitude établie pour nommer les cases d'un tableau consiste à nommer d'abord la ligne puis la colonne.



Cette habitude n'est pas bien compatible avec celle qui s'est établie pour le repérage d'un point par deux coordonnées .



(Ici, on nomme d'abord la coordonnée horizontale qui fixe la position verticale du point.)

C'est cette deuxième habitude qui a prévalu pour la position d'un point ou d'un caractère sur l'écran.

Un bon conseil : lorsqu'on vous parle de (A,B), demandez l'interprétation du premier et du second terme.



Compléments, activités et expériences

Somme, tri...

Pour faire la somme des éléments d'un tableau numérique

$$T(0) + T(1) + T(2) + \dots + T(N),$$

il y a une petite astuce : la variable "réservoir".

On initialise le réservoir R à zéro ; puis on y déverse successivement T (0), T (1), ... comme ceci :

```
10 R = 0 'INITIALISATION
```

```
20 FOR I = 1 TO N
```

```
30 R = R + T (I) 'ACCUMULATION
```

```
40 NEXT I
```

```
50 PRINT "LA SOMME DES ELEMENTS DU TABLEAU  
VAUT :"; R
```

Pour trouver le plus petit des éléments du tableau T, il y a une astuce analogue : “*le minus provisoire*”.

On commence par choisir un minus provisoire : T (0) par exemple ; puis on le compare à chacun des éléments du tableau : dès qu'on en rencontre un plus petit que lui, c'est celui-là qui devient le minus provisoire, et ainsi de suite.

```
100 MINUS = T (0) : IMINUS = 0
110 FOR I = 1 TO N
111 IF T (I) < MINUS THEN MINUS = T (I) : IMINUS = I
112 NEXT I
120 PRINT “LE PLUS PETIT DES ELEMENTS DU TABLEAU
EST :” ; MINUS
130 PRINT “SON NUMERO EST :” ; IMINUS
```

Pour trier les éléments du tableau T et les ranger par ordre de grandeur, il suffit de recommencer le programme précédent avec les éléments restants jusqu'à ce qu'il ne reste plus qu'un élément (ce n'est pas la méthode la plus rapide mais tant pis !).

Pour commencer, transformons le programme précédent en un sous-programme :

```
99 'TROUVER LE MINUS
199 'RETURN
```

Puis décidons que le tableau contenant les éléments de T, rangés dans l'ordre croissant, s'appellera TRANGE. TRANGE(0) sera le plus petit et TRANGE (N) le plus grand :

```
200 'PROGRAMME DE TRI
210 FOR I = 0 TO N
220 GOSUB 99
230 TRANGE (I) = MINUS
240 T (IMINUS) = 999999
250 NEXT I
299 END
```

(L'instruction 240 assure la “disparition” du plus petit élément de T afin de permettre le choix futur du plus petit... après lui).



Remarque

Les programmes précédents permettent aussi de ranger des mots par ordre alphabétique. En effet les comparaisons ($<$, $>$, $=$) ont aussi un sens “alphabétique”.

Il suffit de remplacer les noms de variables et de tableaux numériques par des noms de variables et de tableaux alphanumériques.

Lecture de DATA

C'est avec l'apparition des tableaux que les listes de DATA prennent leur intérêt. En effet, une liste non numérotée est assez fastidieuse à traiter.

Pour parcourir une liste de données, il existe deux méthodes suffisamment caractéristiques pour s'y attarder un instant :

— Ou bien on connaît le nombre d'éléments de la liste. Il faut alors compter les éléments au fur et à mesure de leur “passage”.

— Ou bien on ne connaît pas le nombre d'éléments (parce qu'on ne peut pas ou qu'on ne veut pas le donner). Il faut alors convenir d'un élément particulier indiquant la fin de la liste.

1^{er} cas : le nombre d'éléments de la liste de données est donné en tête de la liste.

```
10 DATA 5, JEAN, LUC, PIERRE, ANDRE, CHARLES
20 READ N
30 FOR I = 1 TO N : READ T$(I) : NEXT I
40 ...
```

Dans ce cas là, lorsqu'on veut ajouter un nom à la liste, il faut remplacer 5 par 6 et ajouter ce nom.

2^{ème} cas : on convient que le mot “Z” termine la liste.

```
10 DATA JEAN, LUC, PIERRE, ANDRE, CHARLES, Z
20 I = 1
30 READ T$(I) : IF T$(I) = "Z" THEN GOTO 40 ELSE I = I + 1 :
   GOTO 30
40 ...
```

Dans ce cas là, lorsqu'on veut ajouter un nom à la liste, il suffit de le placer dans la liste (en tête, par exemple).

D'autre part le comptage des éléments est ici fait par le programme. Après l'instruction 40, le nombre d'éléments de T\$ est exactement I - 1.

Affectation de tableaux

Sur le TO7(-70) et le MO5, il n'est pas possible d'écrire une instruction du genre :

T = S

... lorsque T et S sont des noms de tableaux.

On est en effet obligé d'écrire (si N est le nombre maximum de leurs éléments) :

FOR I = 0 TO N : T (I) = S (I) : NEXT I

De même, pour annuler tous les éléments d'un tableau, on est obligé de commander une boucle, ou des boucles imbriquées pour des tableaux à plusieurs indices.

FOR I = 0 TO N : FOR J = 0 TO M : R (I,J) = 0 : NEXT J : NEXT I



Remarque

Pour "parcourir" un tableau Q à trois indices, ici notés I,J,K, il faut imbriquer trois boucles :

```
FOR I = 0 TO MAXI  
FOR J = 0 TO MAXJ  
FOR K = 0 TO MAXK
```

On fait quelque chose avec Q (I,J,K)

```
NEXT K  
NEXT J  
NEXT I
```

Tracé d'un polygone

Soit à tracer sur l'écran un polygone à N côtés dont on connaît les coordonnées des N sommets :

$(X_1, Y_1) (X_2, Y_2) (X_n, Y_n)$

10 'Lecture des coordonnées

20 DATA 5, 100, 100, 100, 50, 50, 100, 150, 150, 150, 100

30 READ N

40 FOR I=1 TO N : READ X (I), Y (I) : NEXT N

Pour tracer ce polygone sur l'écran, il ne faut pas oublier de tracer le dernier côté qui joint le sommet numéro N au sommet numéro 1.

Exemple

100 'Tracé

110 FOR I=1 TO N-1

120 LINE (X(I),Y(I))-(X(I+1),Y(I+1))

130 NEXT I

140 LINE (X(N),Y(N))-(X(1),Y(1))

Mais il y a une petite astuce pour éviter d'écrire l'instruction "supplémentaire" 20040 : c'est de donner *un double nom* au dernier sommet en l'appelant aussi "numéro 0":

Le tracé est alors plus simple (ainsi d'ailleurs que d'autres traitements que vous pourriez vouloir faire subir à ce polygone).

Voici donc un programme de tracé plus "astucieux".

105 X (0) = X (N) : Y (0) = Y (N)

110 FOR I=0 TO N

120 LINE (X(I),Y(I))-(X(I+1),Y(I+1))

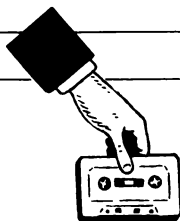
130 NEXT I

(Cette utilisation d'un nom "supplémentaire" est à rapprocher de l'échange de valeurs dans le chapitre 3.)

CHAPITRE 12

Fonctions numériques

Suivi de la cassette



En BASIC, les fonctions mathématiques s'écrivent toujours avec trois lettres.

Si "FON" est le nom d'une fonction et si X est un nombre, la valeur de la fonction est le nombre :

FON (X)



Remarque

En BASIC - MO5, L'utilisateur ne peut pas définir ses propres fonctions.

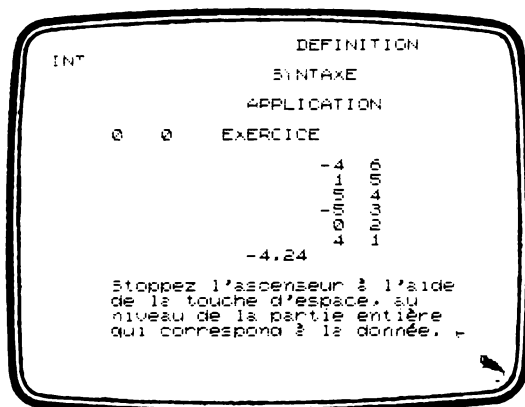
Vous trouverez la liste des fonctions disponibles dans le manuel d'utilisation.

Observez bien les "applications" : elles vous montrent quelques valeurs des fonctions étudiées qui devraient suffire à votre information.

INT

"INT" est l'abréviation de l'anglais "INTEger" qui signifie "entier".

EXERCICE



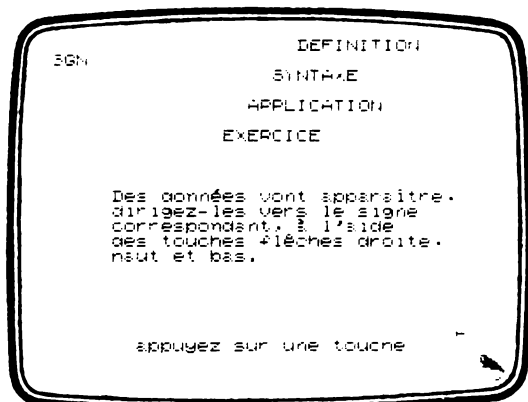
Un ascenseur monte sur "l'immeuble" de droite.

Ici, la partie entière de $-4,24$ valant -5 , il faut arrêter l'ascenseur au 3^{ème} étage.

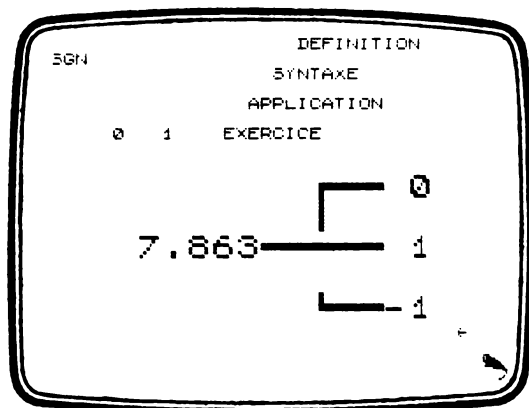
SGN

"SGN" est l'abréviation de "SiGNe".

EXERCICE



Après la lecture de la première page, appuyez sur une touche.



Ici, 7.863 étant positif il vous suffit de ne rien faire. Ou alors appuyez sur la touche \square qui ne changera rien.

Si le nombre à droite avait été négatif, il vous aurait fallu appuyer sur \square .

Attention

Les trois chiffres 0, 1 et -1, écrits à droite, changent de place à chaque nouveau nombre !

RND

“RND” est l’abréviation de l’anglais “RaNDom” qui signifie “hasard”.

Pour bien utiliser RND il faut bien comprendre cette suite de calculs :

RND est un nombre compris entre 0 et 1 (1 exclu).

$K \cdot \text{RND}$ est un nombre compris entre 0 et K (K exclu).

$\text{INT}(K \cdot \text{RND})$ est l’un des nombres entiers : 0, 1, ... ou $K - 1$.

$\text{INT}(K \cdot \text{RND}) + 1$ est l’un des nombres entiers compris entre 1 et K (K compris).

Autres fonctions

TAN (A) est habituellement noté en France, tg (A). (Mais l'habitude est en train de changer !)

EXP (B) est noté, sur une feuille : e^B

LOG (n) est le logarithme népérien de n parfois noté Log (n) ou même L (n)

ABS (n) est habituellement noté : $|n|$

SQR (n) est noté, sur une feuille \sqrt{n}



Un nombre est-il entier ?

Si le nom du nombre est X, la réponse à cette question est la même que la réponse à :

“X est-il égal à $\text{INT}(X)$ ” ?

La phrase...

“Si X est entier faire ceci, sinon faire cela”

... se traduit donc, en BASIC, par :

IF X = INT (X) THEN ceci ELSE cela.

De même pour tester si un nombre A est multiple de B ou non, on demande si A/B est entier.

La phrase...

“Si A est multiple de B faire ceci, sinon faire cela”

... se traduit donc, en BASIC, par :

IF (A/B) = INT (A/B) THEN ceci ELSE cela

... mais aussi par :

IF A MOD B = 0 THEN ceci ELSE cela

Les chiffres d'un nombre

Soit un nombre N, par exemple $N = 1789$.

Il arrive parfois que l'on veuille connaître les chiffres de ce nombre (“les desseins du programmeur sont impénétrables !”).

Quelles instructions écrire de manière à ce que leur exécution donne respectivement à C(0), C(1), C(2) et C(3) les valeurs 9, 8, 7 et 1 (chiffres des unités, des dizaines, des centaines et des milliers de N) ?

Ici le chiffre des milliers est le plus facile à connaître. En effet, c'est le quotient entier de la division de 1789 par 1000.

Ensuite, pour connaître le chiffre des centaines il faut :

— enlever “ses” milliers de 1789 ; résultat : 789

— et faire le quotient entier de 789 par 100.

... Ainsi de suite.

En supposant que les nombres traités ne dépassent pas le million, on écrit donc le programme :

```
10 INPUT N
```

```
20 FOR I=5 TO 0 STEP -1
```

```
21 C (I)= N @ (10I I)
```

```
22 N=N - C (I)*(10I I)
```

```
23 NEXT I
```

```
30 'Vérification
```

```
31 FOR I=5 TO 0 STEP -1 : ? C (I) ; : NEXT I
```

(Voir, dans le chapitre 13, le complément “Afficher des chiffres serrés”.)

Choisir au hasard certaines listes de données

Comment faire lire l’une des trois listes de données suivantes au hasard ?

```
10 DATA 3, 100, 50, 200, 50, 300, 100
```

```
11 DATA 4, 50, 50, 100, 50, 100, 100, 50, 100
```

```
12 DATA 6, 100, 50, 200, 50, 300, 100, 200, 150, 100, 150, 0, 100
```

(Utilisées dans le programme “Dessin d’un polygone” du chapitre 11, ces données feront tracer un triangle, un carré ou un hexagone selon que l’on choisira de lire 10, 11 ou 12.)

Pour résoudre ce problème, nous avons besoin de la fonction RND pour le choix au hasard et de l’instruction RESTORE pour positionner le pointeur de données. Voici une réponse possible :

```
20 I=INT (RND*3)+1 : ON I GOTO 21, 22, 23
```

```
21 RESTORE 10 : GOTO 20
```

```
22 RESTORE 11 : GOTO 20
```

```
23 RESTORE 12 : GOTO 20
```

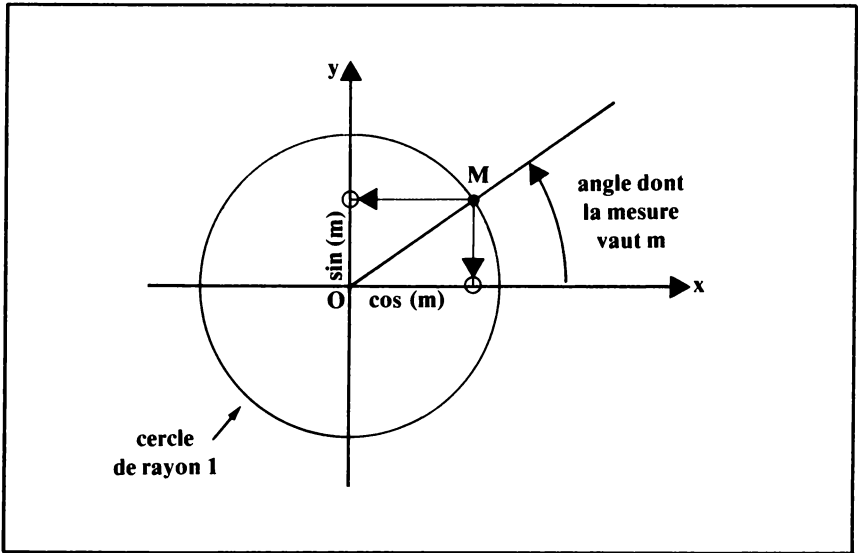
```
30 READ N
```

```
40 FOR I=1 TO N : READ X (I),Y (I) : NEXT N
```


Des cercles

Il n'y a pas, sur votre MO5 ou votre TO7(-70) d'instructions commandant le tracé d'un cercle de centre et de rayon donnés. C'est à vous (et donc à nous) de vous fabriquer cette instruction.

Pour comprendre les formules qui permettent de tracer un cercle il n'est pas vraiment nécessaire d'avoir fait des mathématiques et en particulier de la trigonométrie. Il suffit en effet de savoir ceci :



Dessiner un cercle de centre O de rayon 1 et deux diamètres perpendiculaires.

Le problème est de savoir où se projette le point d'un cercle sur ces deux diamètres pris comme axes de coordonnées ?

La réponse est très simple :

Si on appelle m la mesure de l'angle que fait OM avec l'axe horizontal, alors les coordonnées de M sont :

$\cos (m)$ et $\sin (m)$.

Autrement dit : \cos et \sin sont les fonctions qui font passer de l'angle \widehat{xOM} aux coordonnées de M.

Pour tracer un cercle, il suffit alors de tracer 36 petits segments de droite joignant un point à un autre du cercle (et vous connaissez les coordonnées de ces 36 points). Voici donc le programme traçant un cercle dont le centre a pour coordonnées (A,B) sur l'écran et dont le rayon vaut R (avec les unités de l'écran) :

```
10 CLS : INPUT A,B,R,  
20 PSET (A + R,B)  
30 FOR M=0 TO 360 STEP 10  
40 LINE - (A + R*COS(M), B + R*SIN(M))  
50 NEXT M
```

Des "polygones"

Quelques modifications du programme précédent engendrent une fantastique variété de dessins sur l'écran (dessins que j'ai décidé, en 1972 d'appeler des "jolygones").

Il suffit pour cela :

— de rendre variable de pas de parcours du cercle (H à la place de 10).

— De laisser à l'utilisateur la possibilité de diminuer le rayon du cercle au cours de son tracé !

Voici les modifications à apporter :

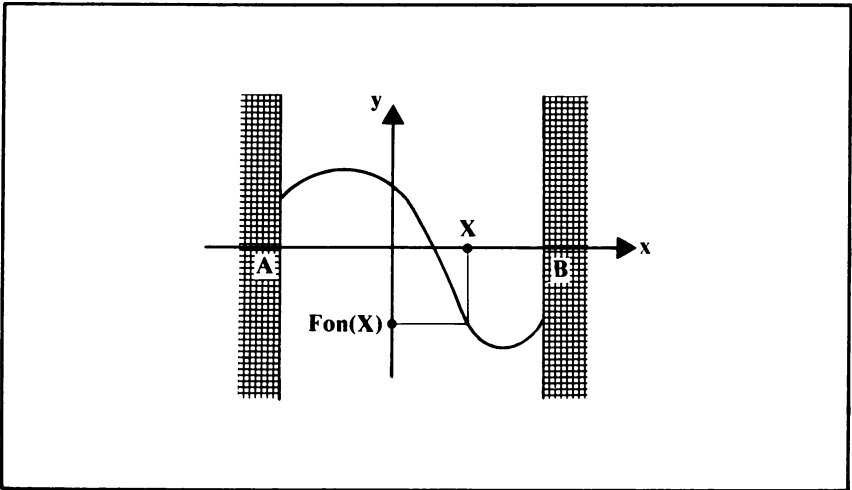
```
10 CLS : INPUT H,K : A = 150 : B = 100 : R = 80  
30 M = 0  
45 R = K*R  
50 M = M + H : IF (R < 8) OR ((K = 1) AND (M MOD 360 <> 0))  
   THEN 40
```

Tracé de courbes

Le seul problème (mais il est de taille) se posant au programmeur qui veut tracer des courbes sur un écran, est celui des unités sur les axes de coordonnées.

En effet, il ne semble pas difficile de marquer les N points représentatifs de la fonction FON sur l'intervalle $[A,B]$:

```
10 INPUT A, B, N
20 FOR X=A TO B STEP (B-A)/N
30 Y=FON(X)
40 PSET(X,Y)
50 NEXT X
```



Seulement voilà, ce programme ne donnera rien sur votre machine.

En effet, les coordonnées en usage sur l'écran n'ont rien à voir avec celles que vous trouverez plus pratiques. En effet :

- Vous voulez que X varie de A à B et non pas de 0 à 320 .
- L'axe des ordonnées de l'écran est dirigé vers le bas, ce qui est contraire à l'usage [voir chapitre 11, paragraphe : $A(i)$, $A\$(j)$].

Conformément aux conseils donnés au chapitre 10 (“Le travail en équipe”), vous avez tout intérêt à “sous-traiter” la transformation des coordonnées grâce à un sous-programme calculant les coordonnées-machines (XM, YM) correspondant à “vos” coordonnées (X,Y).

L’instruction 40 doit donc devenir :

```
40 GOSUB 100 : PSET (XM, YM)
```

Avec :

```
100 'TRANSFORMATION DES COORDONNEES
```

```
110 XM = (X - A)*320/(B - A)
```

```
120 YM = -(Y - A)*200/(B - A)
```

```
199 RETURN
```

ACTIVITÉS

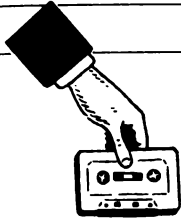
Avec ce sous-programme, les unités sont les mêmes sur les axes vertical et horizontal. Si vous ne voulez pas qu’il en soit ainsi, c’est à vous de faire le travail. Par exemple, écrivez ce qu’il faut pour que l’unité sur l’axe vertical soit K fois plus grande que sur l’axe horizontal !

Il vous reste à remplacer l’instruction 20 par le calcul de la fonction de votre choix pour apprécier les délices d’un juste tracé de courbe.

CHAPITRE 13

Fonctions alphanumériques

Suivi de la cassette



Une fonction numérique a pour argument un nombre et lui fait correspondre un autre nombre.

	un nombre (argument)	FONCTION NUMÉRIQUE	un nombre (résultat)
Exemple	9	SQR	3
	2.3	INT	2
	30	SIN	0.5

Les fonctions “alphanumériques” sont de trois types :

- Celles qui à un nombre font correspondre une chaîne.
- Celles qui à une chaîne font correspondre une chaîne.
- Celles qui à une chaîne font correspondre un nombre.

1^{er} type :

un nombre	→	une chaîne
143	→	“143”
65	→	“A”

2^{ème} type :

une chaîne	et un nombre	→	une chaîne
“BASIC”	3	→	“BAS”
“BASIC”	3	→	“SIC”

3 ^{ème} type :	une chaîne	→	un nombre
	“143”	→	143
	“A”	→	65
	“BASIC”	→	5

Remarquez que le nom de la fonction indique si le résultat est une chaîne ou un nombre (par son dernier caractère : un \$ ou non). Rien n’indique par contre le type de l’argument et le nombre des arguments (c’est à vous de vous en souvenir !).

LEFT\$ RIGHT\$ MID\$

“LEFT\$” signifie “à gauche”.

“RIGHT\$” signifie “à droite”.

“MID\$” est l’abréviation de “MIDdle” qui signifie “au milieu”.

Pour vous souvenir de ce que font ces fonctions vous pouvez penser à une “extension” de leur écriture.

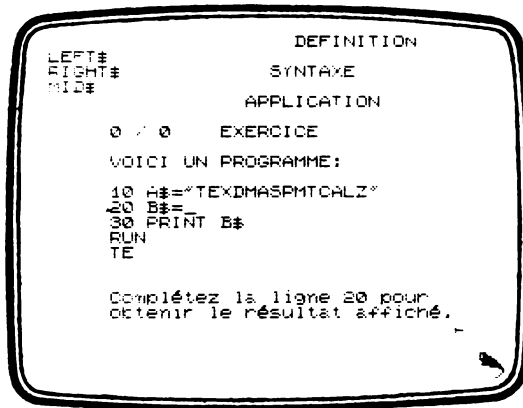
Comme ceci par exemple :

chaîne extraite à gauche de A\$: n caractères
LEFT\$ (A\$, n)

chaîne extraite à droite de A\$: n caractères
RIGHT\$ (A\$, n)

chaîne extraite au milieu de A\$, à partir du i ^{ème} caractère : n caractères
MID\$ (A\$, i , n)

EXERCICE



Vous devez retrouver le morceau de A\$ (ligne 10) qui est affiché et en déduire la fonction utilisée en ligne 20.

Ici "TE" est sur la gauche de A\$; il faudrait donc frapper :

LEFT\$ (A\$,2) **ENTREE** .

VAL STR\$

"VAL" est l'abréviation de "valeur".

"STR" est l'abréviation anglaise de "STRing" qui signifie "chaîne (de caractères)".

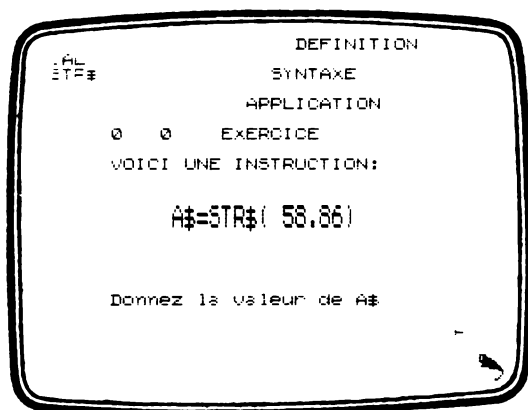
"7" est le chiffre sept

7 est le nombre sept

VAL ("7") vaut 7

STR\$ (7) vaut "7"

EXERCICE



Attention, A\$ est ici une chaîne ; n’oubliez pas les guillemets ! (N’oubliez pas non plus, l’espace remplaçant le signe +.)

Il faudrait donc frapper :

“ 58,86” **ENTREE**

LEN

“LEN” est l’abréviation anglaise de “LENgth” qui signifie “longueur”.

ASC CHR\$

“ASC” est l’abréviation de “ASCII” qui vient de “American Standard Code for Information Interchange”.

“CHR\$” est l’abréviation de “CHaRacter” qui signifie “caractère”.

La correspondance entre un caractère et son code est donnée en annexe.

:	<u>CHR\$</u>	:	
64	→	@	
65		A	
66		B	
67		C	
:	←	<u>ASC</u>	:

Attention

Les codes compris entre 0 et 31 sont réservés à des caractères spéciaux.

Par exemple : CHR\$ (9) vaut **■**
CHR\$ (13) vaut **ENTREE** .

INSTR

“INSTR” est l’abréviation de “IN STRing?” qui signifie “dans la chaîne ?”.

Attention

La fonction INSTR a trois arguments (dont un facultatif) :

- Une chaîne dans laquelle on cherche quelque chose.
- Une autre chaîne qui est recherchée dans la première.
- Un nombre qui indique à partir de quel caractère de la première chaîne on doit chercher la deuxième. (Cet argument est facultatif.)

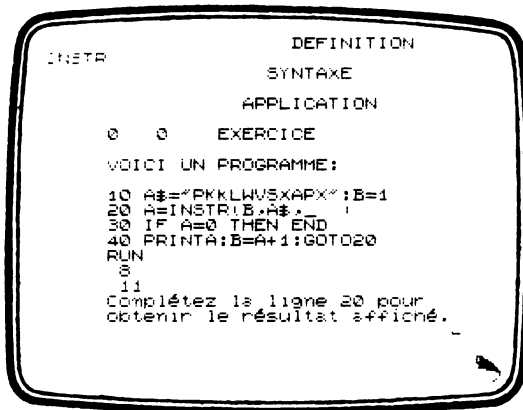
Le numéro du caractère qui, à partir du $i^{\text{ème}}$ caractère de A\$, début la chaîne B\$.
--

INSTR (i , A\$, B\$)

Exemples

- INSTR (“BASIC”, “E”) vaut 0 car “E” n’existe pas dans la chaîne “BASIC”.
- INSTR (“BASIC”, “I”) vaut 4 car “I” est le 4^{ème} caractère de “BASIC”.
- INSTR (“BASIC”, “AI”) vaut 0 car “AI” n’est pas contenu, tel quel, dans “BASIC”.
- INSTR (“BASIC”, “AS”) vaut 2.
- INSTR (“BANANE”, “A”) vaut 2.
- INSTR (3, “BANANE”, “A”) vaut 4.
- INSTR (5, “BANANE”, “A”) vaut 0.

EXERCICE



Le programme donné examine la chaîne A\$ et y recherche toutes les positions du caractère nommé comme 3^{ème} argument dans l’instruction 20.

“II” commence à rechercher ce caractère à partir de la première position (B = 1) ; s’il ne le trouve pas, c’est fini ; sinon, il l’a trouvé en position A et il continue à le rechercher à partir de la position suivante (B = A + 1)...

Ici, le caractère cherché se trouve en position 8 et 11.

Il faudrait donc frapper :

“ X ” ENTREE





Le verlan

Parler le verlan, c'est parler à l'envers !

Précisément, dans chaque mot l'ordre des syllabes est inversé.

Manzessipré, dan qeucha mot, drelor des beullassy ett séverin.

Malheureusement les syllabes ne sont pas faciles à reconnaître dans un mot (par la machine !) ; c'est pourquoi nous nous contenterons d'inverser les lettres d'un mot :

```
TOM$ = " " : FOR I = LEN (MOT$) TO 1 STEP - 1 :  
TOM$ = TOM$ + MID$ (MOT$,I,1) : NEXT I
```

SÉTIVITCA :

1. zevirce nu emmargorp iuq essaf icec !
2. ceci fasse qui programme un écrivez !
3. icec essaf iuq emmargorp nu zevircé !



Remarque

La I^{ème} lettre du mot MOT\$ s'appelle :

MID\$ (MOT\$, I, 1).

Détruire et construire des mots

Les fonctions alphanumériques sont de magnifiques outils "oulipiques" qui vous permettent de traiter ou de maltraiter les mots ou les phrases, à votre convenance.

Les programmes qui suivent vous aideront dans votre entreprise.

Voici d'abord deux sous-programmes :

```
100 'suppression de S$ dans A$
110 N = INSTR (A$, S$)
120 IF N = 0 THEN RETURN
130 A$ = LEFT (A$, N - 1) + MID$ (A$, N + LEN (S$))
199 GOTO 110
200 'insertion de T$ dans A$ à partir de la position P
210 A$ = LEFT (A$, P - 1) + T$ + MID$ (A$, P)
299 RETURN
```

ACTIVITÉS

Avec l'aide de ces sous-programmes, vous devriez arriver à :

— Écrire un programme qui détruit les voyelles d'un mot. "BASIC" devient "BSC".

— Écrire un programme qui écrit un mot en Javanais : "BASIC" devient "BAVASAVIC" (dès qu'une voyelle suit une consonne, on insère les deux lettres "AV").



Remarque

Pour répéter un même travail avec chaque chaîne d'une certaine liste, il faut sûrement écrire un morceau de programme du genre :

```
300 DATA 6, A, E, I, O, U, Y
310 READ N : FOR I = 1 TO N : READ L$ (I) : NEXT I
```

ACTIVITÉS

En sous-traitant à quelqu'un (vous-même ?) un sous-programme qui extrait les mots d'une phrase en repérant les espaces, vous pourrez sûrement généraliser les programmes précédents à des textes.

INPUT	Affichage
BON SANG ! MAIS C'EST BIEN SUR	BN SNG ! MS C'ST BN SR
LA JAVA C'EST ÇA QUI M'VA	LAVA JAVAVAVA C'AVEST ÇAVA QUAVI M'VAVA

Les caractères cachés

Certains caractères peuvent être affichés sur votre écran bien qu'ils ne correspondent à aucune touche du clavier. Le constructeur du clavier a pensé qu'ils n'étaient pas suffisamment utilisés pour mériter une touche. Mais ils ont un code ASCII grâce auquel on peut les "voir".

EXPÉRIENCES

```
10 FOR CODE = 91 TO 96 : PRINT CHR$(CODE) : NEXT CODE
20 FOR CODE = 123 TO 127 : PRINT CHR$(CODE) : NEXT
   CODE
```

Afficher des chiffres serrés

Vous savez que le MO5 ou le TO7(-70) affiche les nombres positifs en les séparant par deux espaces. Pour éviter ces espaces (si on le désire) il faut transformer le nombre en une suite de caractères avant de l'afficher. La fonction STR\$ est là pour ça :

EXPÉRIENCE

```
10 INPUT A, B, C
20 A$ = STR$(A) : B$ = STR$(B) : C$ = STR$(C)
30 PRINT A ; B ; C
40 PRINT A$ ; B$ ; C$
```

Mais il y a aussi la place du signe !

```
50 AA$ = MID$(A$,2) : BB$ = MID$(B$,2) : CC$ = MID$(C$,2)
60 PRINT AA$, BB$, CC$
```

(Comme vous le voyez, le 3^{ème} argument de MID\$ est facultatif et son omission fait extraire tous les caractères à partir de la position indiquée.)



Remarque

Il arrive alors souvent que les nombres à afficher n'aient qu'un seul caractère ; par exemple, on a trouvé les chiffres C (0), C (1), C (2)... d'un nombre et on veut afficher ce nombre (sans séparation entre ses chiffres, évidemment !).

Pour les mathématiciens, on peut aussi "reconstruire" le nombre. Par exemple comme ceci :

```
100 N = 0
110 FOR I = 0 TO 5
120 N = N + C(I) * (10 ↑ I)
130 NEXT I
```

Entrée sans INPUT

Lorsque vous faites un programme devant être utilisé par d'autres, vous pouvez regretter une chose.

Dès que vous demandez à l'utilisateur de frapper une donnée ou une réponse au clavier, il "a la main" et il peut donc frapper sur des touches qui provoquent des actions ou des erreurs non prévues (**RAZ** , **⏏** au lieu de **⏎**...).

Vous pouvez éviter cet inconvénient en gardant le contrôle de l'exécution et en surveillant tout ce qu'il frappe. Pour cela il ne vous faut pas utiliser INPUT mais des suites de INKEY\$.

Le programme que nous vous donnons...

... n'accepte que les frappes de lettres (ligne 15)

... efface les frappes qui ne sont pas des lettres (ligne 16)

... arrête la lecture lorsque l'utilisateur (ligne 14) frappe sur **ENTREE** .

```
10 'Editeur d'entrée de lettres
11 MOT$ = ""
12 F$ = INKEY$ : IF F$ = "" THEN 12
13 F = ASC (F$)
14 IF F = 13 THEN RETURN
```

```
15 IF (F > 64 AND F < 91) OR  
    (F > 96 AND F < 123) THEN MOT$ = MOT$ + F$ : GOTO 12  
16 PRINT CHR$(8) + "  " + CHR$(8) ; : GOTO 12
```

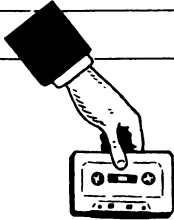
ACTIVITÉS

1. Réaliser un “éditeur d’entrée” qui n’accepte que les lettres, chiffres ou signes en usage dans les textes “littéraires” français.
2. Réaliser un éditeur d’entrée qui n’accepte que les écritures de nombres décimaux (et qui accepte la virgule au lieu du point).

CHAPITRE 14

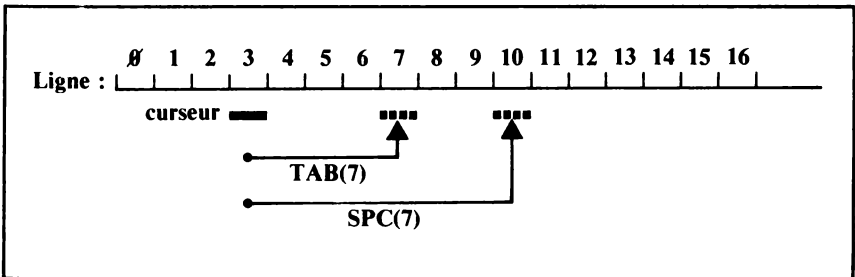
Disposition des résultats

Suivi de la cassette



La présentation des résultats est aussi importante sur un écran que sur une feuille de papier. Les instructions ici étudiées vous permettront de bien choisir et préciser vos “formats” d’affichages c’est-à-dire la “forme” que vous voulez leur donner.

TAB SPC



“TAB” est une abréviation de “TABulation”.

C’est un terme provenant de la frappe sur machine à écrire et consistant à placer automatiquement la “boule” sur une position donnée à l’avance sur la ligne d’écriture.

“SPC” est une abréviation de “eSPaCe”.

PRINT USING

“PRINT USING...” peut se traduire par “Imprimer selon l’usage...”

Les deux instructions...

A\$ = “# # # . # #”

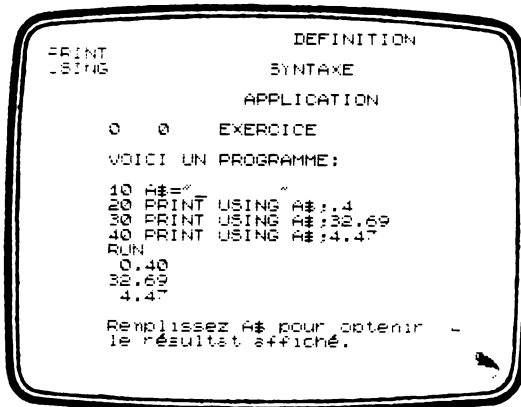
PRINT USING A\$; A, B

... pourraient se lire, en français :

“Imprimer les nombres A et B selon l’usage # # # . # #” (c’est-à-dire avec 3 chiffres avant la virgule et 2 chiffres après).

Comme le montre le programme d’“application”, si le nombre de chiffres avant la virgule est supérieur à 3, il sera (mal) affiché mais précédé du signe %.

EXERCICE



```
DEFINITION
PRINT
USING
BYNTAXE
APPLICATION
0 0 EXERCICE
VOICI UN PROGRAMME:
10 A$="
20 PRINT USING A$;1.4
30 PRINT USING A$;32.69
40 PRINT USING A$;4.47
RUN
0.40
32.69
4.47
Remplissez A$ pour obtenir
le résultat affiché.
```

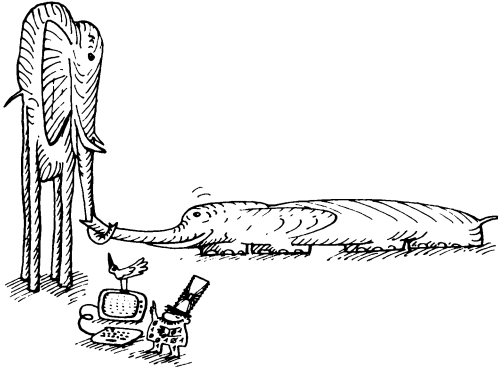
Il s’agit de retrouver le format d’affichage en observant la manière dont chaque nombre est affiché.

Ici, il faudrait frapper :

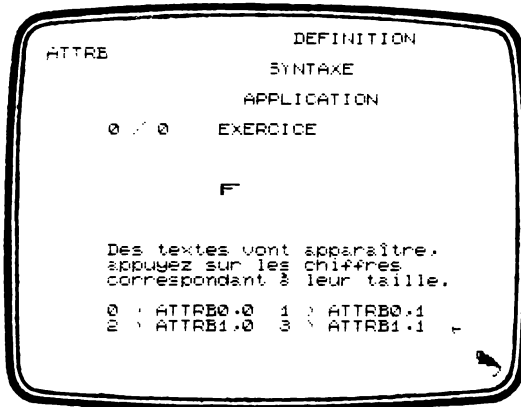
. # # ENTREE

“ATTRB” est l’abréviation de “ATTRiBut” au sens de “qualité” ou “caractéristique”.

En effet la grandeur d’un caractère est un de ses “attributs” de même que sa couleur ou sa forme, ou le fond sur lequel il est écrit.



EXERCICE



Le premier chiffre fixe la largeur, le deuxième fixe la hauteur.

Ici, le F est large (1 d’abord) et de hauteur normale (0 ensuite).

L’instruction convenable est donc ATTRB 1,0.

Il faudrait donc frapper sur **2**.



Tables et tableaux

L'instruction PRINT USING sert beaucoup à présenter correctement des listes ou des tableaux de nombres.

EXPÉRIENCE

```

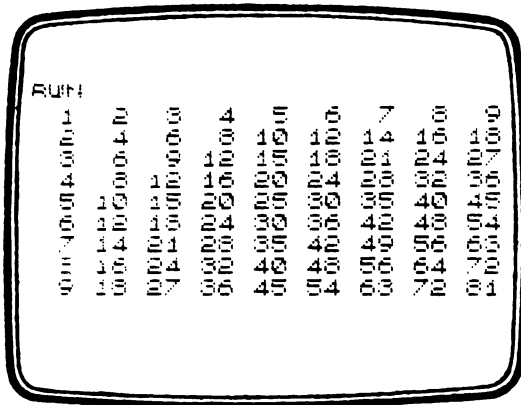
10 INPUT M, N, H, : F$ = "##, #.###, ####, #####,
   ######"
11 FOR X=M TO N STEP H
12 PRINT USING F$ ; X ; SQR (X) ; X ↑ 2 ; X ↑ 3 ; X ↑ 4
13 NEXT X

```

En entrant, par exemple 30, 50, 2, on obtient une belle table des puissances et des racines des nombres pairs entre 30 et 50.

ACTIVITÉS

Voici une "table de Pythagore" que nous avons obtenue sur l'écran.



Pouvez-vous retrouver le programme qui lui a donné naissance ?

Choisir ses caractères

Les caractères normaux sont peu visibles sur l'écran.

Les caractères en double largeur et double hauteur se voient de loin mais c'est un peu "m'as-tu-vu".

Par contre les caractères de largeur normale et de hauteur double ont une certaine élégance.

Voici un programme qui vous permettra de vous faire une idée.

EXPÉRIENCE

```
10 A$ = "Essai du 31 FEVRIER 1985"  
11 FOR I=0 TO 1 : FOR J=0 TO 1  
12 ATTRB I,J : PRINT A$ : ? : ?  
13 NEXT I
```



Remarque

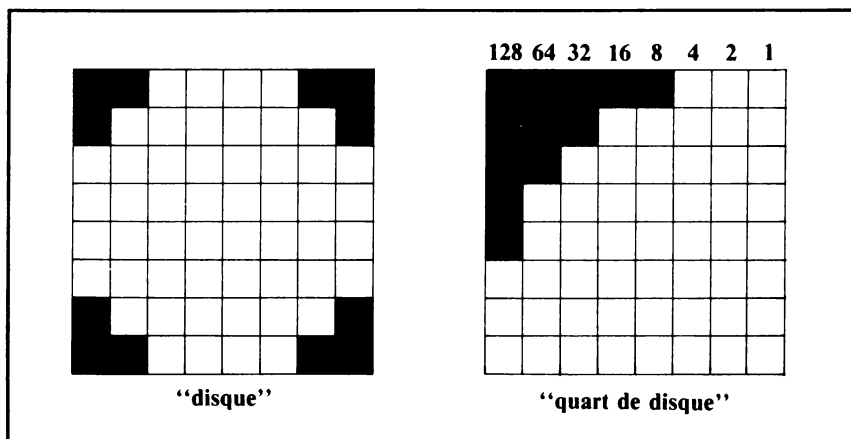
Pour passer à l'écriture en minuscules, il faut appuyer simultanément sur la barre d'espace et la touche "majuscule".

N'hésitez pas à écrire en minuscule ; c'est souvent plus joli !

(Mais n'oubliez pas alors, dans les tests, que la lettre "A", de code 65, est différente de la lettre "a", de code 97.)

Créer vos cercles "pleins"

Le tracé d'un cercle oblige la manipulation des fonctions trigonométriques. Le coloriage de l'intérieur d'un cercle, ligne à ligne, prend énormément de temps. C'est pourquoi, si vous n'avez pas besoin de trop grands cercles, vous pouvez vous les créer par l'instruction DEF GR\$.



Un disque en couleur de fond sera obtenu en dessinant GR\$ (Ø) défini par :

```
DEF GR$ (Ø) = 195, 129, Ø, Ø, Ø, Ø, 129, 195
```

En double hauteur et double largeur cela fait un disque de deux caractères de diamètre.

Mais on peut aussi se "fabriquer" des disques de quatre caractères de diamètre en les dessinant par quart :

EXPÉRIENCE

```
10 CLEAR, 4 'DISQUE
```

```
11 DEF GR$ (Ø) = 248, 224, 192, 128, 128, Ø, Ø, Ø
```

```
12 DEF GR$ (1) = Ø, Ø, Ø, 128, 128, 192, 224, 248
```

```
13 DEF GR$ (2) = 31, 7, 3, 1, 1, Ø, Ø, Ø
```

```
14 DEF GR$ (3) = Ø, Ø, Ø, 1, 1, 3, 7, 31
```

```
15 DISQUE$ = GR$ (Ø) + GR$ (2) + CHR$ (8) + CHR$ (8) + CHR$ (10) + GR$ (1) + GR$ (3)
```



Remarque

Notez l'assemblage des caractères grâce aux codes ASCII des touches de déplacement du curseur.

On arrive ainsi à donner un véritable nom aux dessins (ici DISQUES\$ désigne un disque de 2 caractères de diamètre).

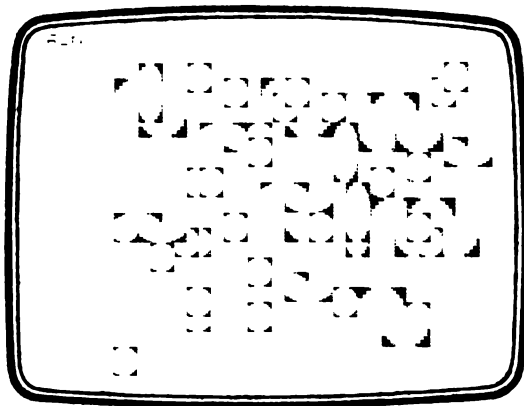
ACTIVITÉS

En vous aidant d'un compas et de papier quadrillé, vous n'aurez sûrement pas trop de mal à dessiner des disques de 3, 4, ... caractères de diamètre (surtout que l'intérieur est formé de caractères "pleins" de code ASCII, 127).

EXPÉRIENCE

L'écran ci-dessous a été obtenu lors d'une exécution du programme suivant associé au précédent :

```
20 SCREEN 1, 0, 0
30 X=5+30*RND : Y=3+21*RND : LOCATE X,Y
40 I=0.5+RND : J=0.5+RND : ATTRB I,J
50 COLOR 0, 1+15*RND
60 PRINT DISQUES$ : GOTO 30
```



ANNEXE 4

CODE ASCII

Code décimal	Caractère ou commande
000	[NUL]
001	[SOH]
002	[STX]
003	[ETX]
004	[EOT]
005	[ENQ]
006	[ACK]
007	[BEL]
008	[BS]
009	[HT]
010	[LF]
011	[VT]
012	[FF]
013	[CR]
014	[SO]
015	[SI]
016	[DLE]
017	[DC1]
018	[DC2]
019	[DC3]
020	[DC4]
021	[NAK]
022	[SYN]
023	[ETB]
024	[CAN]
025	[EM]
026	[SUB]
027	[ESC]
028	[FS]
029	[GS]
030	[RS]
031	[US]
032	SP
033	!

034	“
035	#
036	\$
037	%
038	&
039	'
040	(
041)
042	*
043	+
044	,
045	-
046	.
047	/
048	0
049	1
050	2
051	3
052	4
053	5
054	6
055	7
056	8
057	9
058	::
059	:;
060	<
061	=
062	>
063	?
064	@
065	A
066	B
067	C
068	D
069	E
070	F
071	G
072	H

073	I
074	J
075	K
076	L
077	M
078	N
079	O
080	P
081	Q
082	R
083	S
084	T
085	U
086	V
087	W
088	X
089	Y
090	Z
091	[
092	\
093]
094	↑
095	—
096	—
097	a
098	b
099	c
100	d
101	e
102	f
103	g
104	h
105	i
106	j
107	k
108	l
109	m
110	n
111	o
112	p
113	q
114	r
115	s
116	t

117	u
118	v
119	w
120	x
121	y
122	z
123	[
124	
125]
126	~
127	[DEL]

CARACTÈRES DE CONTRÔLE

- 1 : Nul
- 2 : Début d'en-tête
- 3 : Début de texte
- 4 : Fin de communication
- 5 : Demande
- 6 : Accusé de réception
- 7 : Sommaire
- 8 : Retour arrière du curseur (←)
- 9 : Tabulation horizontale (→)
- 10 : Interligne (↓)
- 11 : Tabulation verticale (↑)
- 12 : Saut de page
- 13 : Retour de chariot
- 14 : Hors code
- 15 : En code
- 16 : Échappement transmission
- 17 : Commande d'appareil auxiliaire 1
- 18 : Commande d'appareil auxiliaire 2
- 19 : Commande d'appareil auxiliaire 3
- 20 : Commande d'appareil auxiliaire 4
- 21 : Accusé de réception négatif
- 22 : Synchronisation
- 23 : Fin de bloc de transmission
- 24 : Annulation
- 25 : Fin de support
- 26 : Substitution
- 27 : Échappement (Escape)
- 28 : Séparateur de fichier
- 29 : Séparateur de groupe
- 30 : Séparateur d'article
- 31 : Séparateur de sous-article
- 32 : Espace

Dans la même collection

Le guide du MO5 — *André Deledicq*

Initiation au Basic TO7/TO7-70 — *Christine et François-Marie Blondel*

Un ordinateur en fête — *Serge Pouts-Lajus*

Un ordinateur et des jeux — *Jean-Pascal Duclos*

Guide pratique de l'enseignement assisté par ordinateur — *Jean-Michel Lefèvre*

Faites vos jeux en assembleur sur TO7/TO7-70 — *Michel Oury*

Jeux vidéo, jeux de demain — *Georges-Marie Becherraz/Alain Graber*

Le Basic DOS du TO7/TO7-70 et du MO5 — *Christine et François-Marie Blondel*

Basic TO7, manuel de référence — *Savena*

Initiation à Logo — *Doris Avram/Michèle Weidenfeld/l'équipe de S.O.L.I.*

Logo, manuel de référence — *D. Avram/T. Savatier/M. Weidenfeld/S.O.L.I.*

Logo, des ailes pour l'esprit — *Horacio C. Reggini*

Initiation au Forth — *S.E.F.I.*

Forth, manuel de référence

Manuel de l'assembleur 6809 du TO7/TO7-70 — *Michel Weissgerber*

Manuel de l'assembleur 6809 du MO5 — *Michel Weissgerber*

La face cachée du TO7/TO7-70 — *Jean-Baptiste Touchard*

Manuel technique du TO7/TO7-70 — *Michel Oury*

Manuel technique du MO5

Macintosh, Multiplan, Macpaint — *Eddie Adamis*

Vous et l'ordinateur APPLE — *Edward H. Carlson*

Écrivons un programme pour APPLE — *Royal Van Horn*

Le Logo sur APPLE — *Harold Abelson*

Premiers pas avec le ZX-SPECTRUM — *Ian Stewart/Robin Jones*

Plus loin avec le ZX-SPECTRUM — *Ian Stewart/Robin Jones*

Le langage machine du ZX-SPECTRUM — *Ian Stewart/Robin Jones*

Guide pratique de l'ORIC-ATMOS du Basic à l'assembleur — *Bussac/Lagoutte*

Des programmes pour votre ORIC — *Michel Piot*

Premiers pas avec le COMMODORE 64 — *Ian Stewart/Robin Jones*

Musique sur COMMODORE 64 — *James Vogel/Nevin B. Scrimshaw*

Guide pratique du VIDÉOTEX et du MINITEL — *Saboureau/Bouché*

Guide pratique de l'ordinateur personnel d'IBM — *Boisgontier/Daloz/Emery/Portefaix/Salzman*



IMPRIMERIE AUBIN, 86240 LIGUGÉ

D.L., mars 1985. — Impr., L 19705

Imprimé en France