Module 18 Introduction à la programmation

Il existe deux manières de se servir du langage BASIC. La première manière s'appelle le mode direct, le mode bureau, le mode immédiat et c'est celle que nous avons pratiquée jusque-là. Les ordres que nous donnions au MO6 de cette façon s'appellent des commandes : ce sont des instructions qui s'exécutent immédiatement après l'appui de la touche ENTREE.
Nous avons beaucoup insisté sur la possibilité offerte en mode direct de ranger des données (de type numérique ou chaîne de caractères) dans la mémoire vive du MO6 et de pouvoir les récupérer à tout moment. Toutes ces manipulations nous ont permis d'introduire et de pratiquer les principales commandes du langage BASIC.
Il est temps de passer à l'étage supérieur de ce bel édifice et d'examiner la deuxième manière d'utiliser le MO6, le programme.
Le principe de fonctionnement d'un programme repose sur la possibilité offerte par le système de ranger les commandes en mémoire en vue d'une exécution différée.
Le stockage des commandes pose des problèmes techniques considérables qu'il n'est pas question d'évoquer ici. Il suffit au programmeur de connaître le procédé qui réalise un tel stockage. En BASIC, il est très simple.
Pour ranger une commande en mémoire, il suffit de la faire précéder d'un nombre entier. Cette commande ainsi rangée s'appelle une ligne d'instruction et le nombre qui la précède est le numéro de cette ligne.

Premier exemple, premier programme

1 PRINT 1515 [ENT]

La seule conclusion intéressante à tirer de ce premier exemple est la suivante : il suffit qu'une commande soit précédée d'un nombre pour que l'exécution n'ait pas lieu. Le nombre 1515 n'a pas été imprimé, le OK n'apparaît pas, la machine attend quelque chose.

Il faut également observer que toutes les commandes examinées dans les modules précédents commencent par une lettre : la raison en apparaît maintenant. Les questions auxquelles nous répondrons par la suite découlent naturellement de cette première expérience
— Comment exécuter cette ligne d'instruction ?
— Comment la modifier ?
— Comment la supprimer ?
— Comment enregistrer d'autres instructions ?
Ces manipulations font appel à des mots du langage BASIC, des commandes particulières qu'il faut bien connaître avant de lancer votre premier programme.

Nous ne ferons pas au lecteur l'injure d'exécuter devant lui un programme comportant une seule instruction. Voici une boucle répartie sur trois instructions. Il eût été possible de la stocker en une seule ligne comme en commande, mais il est intéressant de voir comment la machine gère plusieurs lignes de programme. Ces trois lignes, dans notre programme, sont numérotées 1, 2 et 3. Naturellement.

```
1 FOR I = 50 TO 55
2 PRINT I;
3 NEXT
```

Exécution

Pour exécuter cette superbe boucle, il faut taper RUN (mode direct).

```
RUN
50  51  52  53  54  55
OK
```

RUN est donc une commande de lancement de programme. C'est ici qu'il faut saisir la différence fondamentale entre le mode direct et le mode programme. Chaque ligne d'instruction étant numérotée et stockée en mémoire, chaque ordre RUN lance l'exécution du programme autant de fois que l'utilisateur le désire.

```
RUN
50  51  52  53  54  55
OK
RUN
50  51  52  53  54  55
OK
```

Édition

Si vous recommencez un certain nombre de fois cette manœuvre, il est possible que par un effet de défilement (*scrolling*), toutes les instructions du programme disparaissent de l'écran. Elles n'ont pas pour autant disparu de la mémoire. La commande LIST permet de les rappeler.

```
LIST
1 FOR I=50 TO 55
2 PRINT I :
3 NEXT I
```

Certains aménagements de la commande LIST permettent de consulter (lister) certaines instructions.

— LIST : affichage de toutes les lignes d'instruction du programme.
— LIST N-P : affichage des lignes d'instructions dont les numéros vont de N à P.
— LIST N- : affichage de toutes les lignes d'instructions à partir du numéro N.

Numérotation

Le programmeur peut décider d'introduire ses instructions dans l'ordre qu'il désire mais seul l'ordre des numéros sera pris en considération dans la mémoire. La commande LIST affiche les lignes de programme dans l'ordre de leur numéro.

```
2 A=578
1 B=609
LIST
1 B=609
2 A=578
```

Module 19 Conseils préalables

Numéroter

La remarque précédente concernant l'ordre des instructions montre qu'il est possible d'enrichir le programme à tout moment par de nouvelles lignes. Encore faut-il pouvoir le faire. Dans cette optique, les programmeurs ont pris l'habitude de numéroter les lignes de 10 en 10 afin d'insérer facilement à tout endroit de nouvelles lignes de programme. Le programmeur reste entièrement libre de son choix. Seules contraintes :
— Les numéros de ligne doivent être des nombres entiers.
— Ils ne doivent pas dépasser 63 999.
Un programme peut très bien commencer par la ligne 3142 qui sera la première à être exécutée après RUN.

Modifier

Vous l'apprendrez en programmant, la modification, la correction occupent une grande partie du travail de programmation. Le cas le plus bénin concerne une faute de syntaxe dans une ligne de programme. Si votre programme contient une ligne 640 ainsi enregistrée

```
640 PRIT "BONJOUR"
```

il est probable que tôt ou tard, la commande RUN soit suivie d'un message d'erreur. L'interpréteur BASIC vous signale le numéro de la ligne où l'erreur a été détectée.

```
Syntax Error in 640
```

Pour remédier à ce fâcheux incident tapez la commande LIST suivie d'un point.

```
LIST.
640 PRIT ■ "BONJOUR"
```

Vous ne manquez pas alors de vous écrier : "Bon sang ! Mais c'est bien sûr, j'ai oublié le N de PRINT !"

C'est d'autant plus facile que le carré plein vous indique que l'erreur se situe sur l'instruction qui le précède directement. Amenez le curseur à l'endroit requis (sur le T de PRIT), puis tapez INS suivie de N. Validez cette ligne par ENTRÉE et l'erreur est réparée.
Il faut signaler que les erreurs sont détectées une par une et c'est seulement la première erreur rencontrée qui vous est signalée.

Supprimer

Pour supprimer une ligne de programme, il suffit de taper le numéro de cette ligne suivi de ENTRÉE. Cette nouvelle ligne d'instruction vide écrase la précédente, ce qui revient à la supprimer. Elle ne sera plus affichée par LIST.
On peut aussi utiliser la commande DELETE (supprimer) surtout utile pour supprimer un grand nombre de lignes. Elle suit les mêmes principes que LIST.

— DELETE N-P : suppression des lignes d'instructions dont les numéros vont de N à P.

— DELETE N- : suppression de toutes les lignes du programme à partir de la ligne de numéro N.
Pour supprimer tout un programme, il suffit de taper DELETE 0-.
Bien entendu, DELETE doit être utilisé avec beaucoup plus de circonspection que LIST car la suppression d'une ligne de programme est sans appel : tout regret est inutile.

CLEAR

CLEAR est une instruction qui initialise les variables, 0 pour les variables numériques, vide pour les chaînes de caractères. CLEAR peut également servir à réserver de la place en mémoire pour les variables, suivant leur type. Il est parfois utile de placer cette instruction en début de programme mais, la plupart du temps, la commande RUN a le même effet (initialisation des variables).

NEW

La commande NEW efface tout le programme en mémoire (prudence...). Il est très important de vous en servir au moment où vous faites vos gammes de programmation. Vous vous exercez en changeant souvent de projet, donc de programme. Le MO6 doit être informé que vous changez d'activité afin de ne pas mélanger les lignes de deux programmes différents. Un bon conseil donc : lorsque vous attaquez un nouveau programme, commencez par la commande NEW.
Le BASIC du MO6 accepte l'instruction NEW dans une ligne de programme. Lorsque l'interpréteur passe sur cette ligne, il supprime tout. C'est fini. Programme auto-destructif.

Interrompre

Pourquoi interrompre un programme qui tourne ? Plusieurs raisons peuvent se présenter : le programme vous ennuie, il ne vous donne pas les résultats attendus ou, plus grave, vous sentez ou vous savez qu'il ne s'arrêtera jamais tout seul et qu'une intervention extérieure est nécessaire.
Nous allons prendre l'exemple d'un programme particulièrement ennuyeux puisqu'il s'apprête à écrire 10 000 fois ENNUI, ce qui est fort lassant.

```
10 FOR X=1 TO 10000
20 PRINT "ENNUI"
30 NEXT X
```

Interrompre définitivement
Il existe plusieurs méthodes :
1. Détruire l'appareil (onéreux).
2. Couper le courant (peu élégant).
3. Appuyer sur la touche de réinitialisation RESET (efficace). Sélectionner le BASIC 128 dans la page en-tête.
4. Appuyer ensemble sur les touches CONTROLE (CNT) et C.
Cette dernière méthode est évidemment celle que nous recommandons. La commande clavier CNT-C interrompt le déroulement du programme mais, contrairement à la touche RESET, elle laisse intacts les contenus des variables.

Dans l'exemple précédent, après avoir laissé le programme tourner quelque temps, vous l'interromprez par CNT-C. En exécutant ensuite la commande PRINT X, vous saurez combien de fois le mot ENNUI a été imprimé. Lorqu'un programme est interrompu par CNT-C, le message "Break in" suivi d'un numéro de ligne vous indique où l'interruption est intervenue.

Interrompre provisoirement
La touche STOP interrompt provisoirement le déroulement du programme. Aucun message particulier n'apparaît à l'écran. Tout reste en suspens. L'appui sur une touche quelconque relance la machine. Les interruptions de ce type peuvent être répétées. On peut ainsi figer à volonté un écran qui défile un peu vite. L'instruction STOP peut être utilisée dans un programme pour l'interrompre. Après le message Break, on peut relancer le déroulement en tapant la commande CONT.

Module 20 Sauvegarde d'un programme

Vous n'en êtes pas encore à rédiger de longs programmes mais nul ne peut douter un instant que ça ne tardera pas. Les instructions d'un programme, tout comme les contenus des variables, sont enregistrées en mémoire vive, c'est-à-dire une mémoire volatile qui s'effacera en l'absence d'alimentation électrique. Il faut donc faire appel à un stockage extérieur permanent de type magnétique.
Le MO6 dispose d'un lecteur-enregistreur de programmes (en abrégé LEP), capable de gérer des cassettes.
Conseils pour l'utilisation des cassettes :
— Rembobinez toujours les cassettes et remettez le compteur à zéro. Cela vous permettra de repérer les numéros de début et de fin d'enregistrement.
— Attention : le début de la bande (dite "amorce") est absolument inutilisable pour l'enregistrement.

Pour enregistrer ("sauver") un programme

Commencez donc par en faire un. Nous vous proposons celui-ci qui est d'une grande beauté.

```
10 PRINT "CECI EST UN ESSAI D'ENREGISTREMENT"
```

— Choisissez-lui un nom, par exemple TRUC.
— Appuyez simultanément sur les touches [●] et [►] du LEP.
— Commandez sur le clavier du MO6

```
SAVE "TRUC"
```

Le LEP se met alors à faire avancer la bande de la cassette. Votre programme est en train de s'enregistrer sur la bande. Lorsque la bande s'arrête, le message OK apparaît sur l'écran.
SAVE "TRUC" recopie le contenu de la mémoire vive sur la bande magnétique sous le nom de TRUC. Ce nom servira à relire le programme.

```
SAVE "TRUC"
_
```

Pour charger ("lire") un programme

LOAD
Si vous avez enregistré un programme c'est sans doute que vous voulez le retrouver plus tard (après le dîner). Pour transférer le contenu de la cassette dans la mémoire vive du micro-ordinateur, il faut taper l'instruction LOAD "TRUC" si votre programme a été sauvegardé par SAVE "TRUC".

Attention : c'est à vous de placer la bande au bon endroit en face de la tête de lecture. D'où l'intérêt d'avoir noté le numéro au moment de l'enregistrement.

Le LEP se met alors à faire avancer la bande de la cassette. Le MO6 affiche donc

```
SEARCHING
```

Si d'autres programmes sont enregistrés avant TRUC, ils ne sont pas lus mais simplement "parcourus" et le MO6 affiche leur nom, par exemple :

```
Skip : TOTO      .BAS
```

(*Skip* veut dire "sauté" et BAS est l'abréviation de BASIC). Lorsque la partie où est enregistré le programme TRUC vient devant la tête de lecture, le message suivant est affiché :

```
Found : TRUC     BAS
```

Laissez faire. Votre programme est en train de se charger dans la mémoire. Lorsque la bande s'arrête, le message OK apparaît sur l'écran. Vous pouvez vérifier que TRUC est maintenant présent en mémoire en commandant LIST ou RUN.

```
LOAD "TRUC"
Skip TOTO.BAS
Found TRUC.BAS
OK
_
```

Problème au chargement !

— Si le message

```
Device I/O Error
OK
```

apparaît en cours de lecture ou de chargement, il faut rembobiner la bande avant le début de l'enregistrement pour tenter un nouvel essai de chargement.

MOTORON/MOTOROFF

Le BASIC vous propose deux instructions pour commander le LEP à partir du MO6.
— MOTORON qui provoque le défilement de la bande, à condition que la touche [►] soit enclenchée. Si la touche ENREG est aussi enclenchée, il y a effacement de la bande. Pendant ce défilement, on peut frapper toute commande voulue sur le clavier, en particulier la suivante.
— MOTOROFF qui arrête le défilement de la bande.

Remarque : si vous ne mettez qu'un seul programme par cassette, il n'est plus nécessaire d'indiquer le nom du programme au moment du chargement. Il suffit de faire :

```
LOAD" [ENT]
```

C'est le premier programme présent sur la cassette qui est alors chargé.

Aparté : on range des données en mémoire vive par l'instruction d'affectation (=). On range des instructions en mémoire vive en les faisant précéder d'un numéro d'ordre. On range un programme (ensemble de lignes d'instructions) sur la cassette par SAVE. Il est possible également de ranger des données sur la cassette : c'est un fichier. Ce vaste sujet sera traité dans les modules 47 et 48.

Quelques conseils pour gérer vos cassettes

Lorsque vous avez enregistré quelques-uns de vos programmes et recopié quelques programmes de vos nombreux amis, vous risquez de vous mélanger les bandelettes ! Un seul moyen de vous en sortir : soyez très strict sur les méthodes de classement.

Pour vous aider, voici réunis quelques conseils que, bien sûr, vous ne suivrez pas... avant d'avoir compris que vous ne pouvez pas faire autrement...

— Numérotez ou donnez un nom à vos cassettes et marquez leurs deux côtés. Attention : notez ce nom *sur* la cassette (avec un autocollant blanc) et non sur la boîte (les boîtes sont interchangeables).
— Tenez un cahier où vous notez pour chaque cassette (repérée par son nom) la suite des enregistrements et les numéros-compteur correspondants.
— Rembobinez toujours une cassette avant de la sortir du LEP.
— Enregistrez vos programmes sur deux cassettes, l'une dont vous vous servez, l'autre que vous stockez par précaution.
— Placez vos cassettes dans des boîtes.
— Régulièrement, récupérez les cassettes contenant les enregistrements (de travail) dont vous ne vous servez plus et enregistrez du "blanc".
— Si vous tenez à un programme, enregistrez-le deux fois sur une même cassette et une autre fois sur une cassette à laquelle vous touchez le moins possible.
— Si des erreurs se produisent, gardez votre calme et réfléchissez.

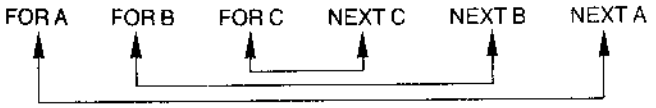
Module 21 Premiers programmes

Nous pouvons reprendre sous forme de programmes les exemples que nous avons traités en mode direct. La possibilité d'écrire sur plusieurs lignes va nous permettre d'emboîter facilement plusieurs boucles : nous arrivons enfin à des résultats saisissants !

Exemple 1 : la tortue dessine de nouveau un octogone !

```
20 CLS
30 TURTLE 0
40 TRACE 1
50 FOR N=1 TO 8
60 FWD 30:HEAD 32
70 NEXT N
```

Exemple 2 : il est possible d'encastrer trois, quatre ou davantage de boucles en respectant toujours scrupuleusement les ordres d'emboîtement.



Rappelons-nous de l'instruction SCREEN.
SCREEN X, Y, Z préparera une écriture de couleur X sur un écran de couleur Y bordé de couleur Z. Pour en voir de toutes les couleurs, on utilisera la recette suivante :
1. Promener le curseur sur tout l'écran en y écrivant n'importe quoi, éventuellement des bêtises.
2. Remonter le curseur en haut à gauche grâce à ↖ .
3. Taper le programme suivant en faisant CNT-X à la fin de chaque ligne et avant ENTREE pour effacer les caractères qui pourraient traîner sur la ligne.

```
10 FOR X=0 TO 15
20 FOR Y=0 TO 15
30 FOR Z=0 TO 15
40 SCREEN X,Y,Z
50 NEXT Z,Y,X
60 SCREEN 4,6,6
```

4. Mettre des lunettes de soleil pour éviter l'éblouissement, puis taper RUN. Le spectacle dura assez longtemps car il passe en revue les 4 096 colorations différentes de l'écran. On observera de temps à autre la disparition de l'écriture lorsque la couleur de l'écriture et celle du fond sont identiques (X = Y).
Nous avons eu la délicatesse d'ajouter la ligne 60 pour ne pas vous laisser dans un épais brouillard orange.

Exemple 3: Votre MO6 vous offre 4 096 teintes différentes, les voici :

```
10 FOR X=4095 TO 0 STEP -1
20 FOR Y=1 TO 10 : LOCATE 0,0 : PRINT X
30 PALETTE 0,X
40 NEXT Y : NEXT X
```

L'instruction PALETTE en ligne 30 affecte à la couleur 0, la teinte X. C'est le fond d'écran, noir au début qui reprendra successivement les 4096 teintes.

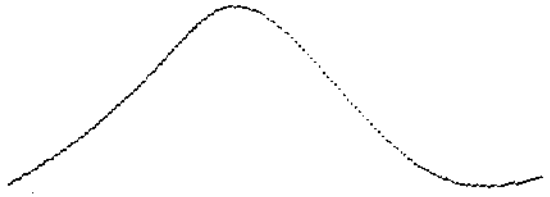
Des courbes pour les exercices mathématiques !

Le tracé des courbes est une bonne occasion d'utiliser l'instruction DEFFN qui permet à l'utilisateur de définir des fonctions particulières. C'est un bon et long travail de programmation que de chercher un programme qui affichera des courbes choisies par l'utilisateur dans n'importe quelle fenêtre. Excellent outil de travail pour les étudiants en mathématiques.
Soit donc à tracer la courbe $f(x)=\frac{2x^2+x-1}{x^2-x+1}$

Voici un petit programme "bricolé" pour afficher cette jolie courbe.

```
10 CLS
20 DEFFN S(X)=(2*X*X+X-1)/(X*X-X+1)
30 EX=50:TX=100:EY=25:TY=50
40 FOR X=-2 TO 2 STEP 1/EX
50 PSET (X*EX+TX,FNS(X)*EY+TY)
60 NEXT X
```

Le résultat obtenu est prometteur...



La difficulté dans ce genre de travail consiste à calculer les coefficients multiplicateurs (EX et EY) de façon à ce que la courbe soit assez grande sans pour autant sortir de l'écran.

Nous reviendrons sur les fonctions mathématiques dans le chapitre 34. Nous espérons que ces premiers exemples de programmes vous séduisent et que vous ne nous quitterez plus d'une ligne.

Attention : ce qui est affiché sur l'écran n'est pas forcément en mémoire et inversement.
On oublie souvent d'effacer une instruction dont on a modifié le numéro. Soit, par exemple, l'instruction affichée

```
7 COLOR 1,2
```

si on désire plutôt donner à cette instruction le numéro 9 (parce qu'elle doit suivre et non précéder l'instruction 8), alors il faut faire successivement ceci :
— Amener le curseur sous le 7 en début de ligne
— Frapper le chiffre 9 et valider en appuyant sur ENTREE.

L'instruction

```
9 COLOR 1,2
```

est alors enregistrée. Attention ! L'instruction 7 n'est plus affichée à l'écran mais elle existe toujours en mémoire. Il faut donc l'effacer en frappant 7 puis en validant. Il n'y a alors plus d'instruction numéro 7 en mémoire, mais il y a une nouvelle instruction numéro 9.

Attention

Pour entrer en mémoire une ligne de programme, il faut frapper sur la touche ENTREE; sinon, l'ordre est écrit sur l'écran mais il n'aura aucune chance d'être exécuté. N'utilisez pas les flèches pour déplacer le curseur d'une ligne à l'autre lorsque vous écrivez un programme. Si jamais vous désobéissez à ce conseil, vous aurez des ennuis qui se traduiront par une erreur de syntaxe qui vous paraîtra mystérieuse. Un conseil : si vous n'arrivez pas à detecter l'erreur signalée par

```
Syntax Error in 50
OK
```

... n'hésitez pas à supprimer l'instruction 50 de la mémoire, puis retapez une nouvelle instruction 50 si vous le jugez nécessaire.

Commentaires

Ecrire des programmes, les enregistrer, les reprendre longtemps après pour les enrichir, c'est l'activité classique du progammeur amateur ou professionnel. Pour assurer la maintenance du logiciel, c'est-à-dire la possibilité d'intervenir dans l'écriture d'un programme après sa réalisation finale, il faut que le programme soit documenté. Avec des programmes courts de diffusion restreinte (cadre familial), on admet que la documentation soit inscrite dans les instructions du programme lui-même, de telle façon que le listing du programme permette à quiconque de s'y retrouver et de comprendre son organisation en vue d'une éventuelle intervention.
Une instruction BASIC permet d'inclure des commentaires dans les programmes : REM. Tout ce qui suit REM dans une ligne d'instructions est ignoré par l'interpréteur BASIC lors de l'exécution mais conservé en mémoire. REM peut être placé en début ou en cours de ligne, mais après REM, aucune instruction ne peut être exécutée sur la même ligne. Votre BASIC admet l'apostrophe (') comme abréviation de REM.

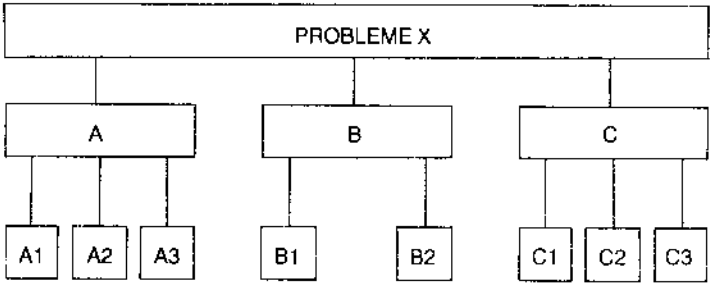
Module 22 Arts programmatiques

Le BASIC n'est pas le seul langage de programmation. Apprendre à programmer en BASIC comme dans tout autre langage devrait se décomposer en deux phases distinctes :
— Apprendre à programmer.
— Apprendre le BASIC.
Si vous négligez complètement la première de ces phases, la plus difficile, mais en même temps la plus importante, vous vous condamnez à n'être qu'un "bidouilleur"... Le qualificatif de bidouilleur est décerné par les vrais informaticiens aux gentils amateurs qui programment plus pour leur propre satisfaction que pour l'amour de l'art programmatique. Le BASIC est le langage chouchou de tous les bidouilleurs, au-delà des frontières et des continents.
Nul ne réconciliera jamais les bidouilleurs et les experts ès programmation mais quitte à se proclamer bidouilleur, autant le faire avec un certain panache, c'est-à-dire sans tout ignorer des arts programmatiques et de leur arme maîtresse, l'analyse descendante.

L'analyse descendante

L'idée de l'analyse descendante est simple et efficace. Programmer c'est résoudre un problème par un algorithme, c'est-à-dire une suite d'opérations élémentaires parfaitement décrites dans leur contenu et dans leur enchaînement. L'analyse descendante mène à un algorithme en respectant les deux principes suivants :
— Décomposer le problème en ses principales phases.
— Traiter chaque partie du problème de la même façon que le problème général.

Pratiquement, l'analyse descendante oblige à toujours s'occuper de l'essentiel en renvoyant à plus tard ce qui peut être considéré comme accessoire. Les Anglo-Saxons appellent cela la "*top down analysis*", c'est-à-dire analyse de haut en bas, et c'est tout à fait clair.
Sur le schéma suivant, le problème X est décomposé en trois tâches A, B et C, elles-mêmes partagées en quelques sous-tâches. On peut continuer jusqu'en bas, c'est-à-dire jusqu'à la prise en compte de tous les détails.



Les principaux avantages offerts par l'analyse descendante sont :
— possibilité de partager entre plusieurs programmeurs,
— examen des difficultés une par une,
— structuration automatique des programmes,
— indépendance vis-à-vis du langage de programmation.

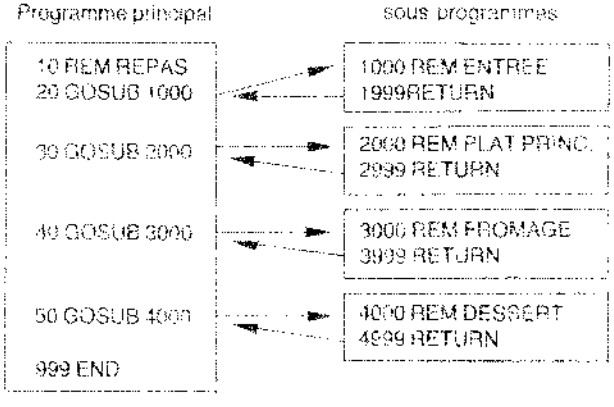
Est-ce à dire que l'analyse descendante doit être mobilisée à chaque fois que l'on doit écrire trois lignes de BASIC ? Certainement pas et il eût été bien pédant de nous en réclamer à tout moment dans ce livre. D'ailleurs, il faut le dire, le BASIC est un langage qui s'accommode assez mal de la programmation structurée. Il existe Pascal qui a été fait pour cela et qui le fait bien. Pourtant les principes de l'analyse descendante doivent encourager le lecteur à respecter en toutes occasions ce commandement :

Réfléchir d'abord,
programmer ensuite

Sous-programmes

Les principes de l'analyse descendante font des sous-programmes l'un des outils privilégiés de la programmation. Les sous-programmes servent à renvoyer à plus tard ce qui ne peut pas être fait immédiatement afin de ne pas se perdre dans les détails.
Un sous-programme se présente comme une suite d'instructions dans le programme. Pour que le sous-programme apparaisse bien

comme tel dans le listing, il est conseillé de commencer la numérotation par un nombre "rond" (100, 200, 1000, 2000, 10000, 20000, etc.) et de placer une remarque sur la première de ces lignes, indiquant la tâche remplie par le sous-programme. La fin du sous-programme est marquée par l'instruction RETURN.
Pour exécuter le sous-programme, il faut utiliser l'instruction GOSUB suivie du numéro de la première ligne du sous-programme que l'on désire appeler. De cette façon, le listing se partage en deux parties :
— le programme principal dans lequel on appelle les sous-programmes,
— l'ensemble des sous-programmes.
Toutes ces explications peuvent paraître un peu ardues au débutant mais il nous semble très important de bien comprendre dès le début la structure d'un programme BASIC. Le programme principal doit se situer en tête du programme ; il commence par une remarque donnant le nom (éventuellement l'auteur) du programme et se termine obligatoirement par l'instruction END marquant la fin du programme principal.
Un dessin valant mieux qu'un long discours, voici un schéma résumant la préparation informatique d'un repas classique. Le rôle du chef cuisinier est tenu par le programme principal qui appelle les sous-programmes correspondant à chaque plat.



Commentaires : l'exécution des sous-programmes se fait sur le mode de l'aller et retour : GOSUB pour aller, RETURN pour revenir. La numérotation réserve des tranches entières. Il est intéressant de fixer les limites du programme et des sous-programmes de cette façon. Le programme principal est entre 1 et 999, les sous-programmes entre 1000 et 1999, entre 2000 et 2999, etc. Si donc vous décidez d'écrire un sous-programme 5000, rien ne vous empêche de commencer par taper

```
5000 REM CAFE
5999 RETURN
```

et de vous laisser le temps de réfléchir pour savoir comment on prépare le café en BASIC.

Module 23 Rupture de séquence

Tant qu'il ne rencontre pas d'instruction GOSUB, le programme se déroule séquentiellement, c'est-à-dire dans l'ordre des numéros de lignes. Rien n'empêche d'appeler les sous-programmes dans un ordre quelconque. Il suffit de changer les instructions du programme principal.

```
20 GOSUB 3000
30 GOSUB 1000
40 GOSUB 4000
50 GOSUB 2000
```

Pour un repas particulièrement indigeste. L'instruction END est la dernière ligne du programme principal juste avant la première ligne du premier sous-programme.

Rien n'empêchera aucun programmeur d'avoir recours à une méthode personnelle qui n'est pas forcément la plus rationnelle ni la plus rigoureuse mais qui peut s'avérer être pour l'intéressé la plus agréable ou la plus efficace.

L'auteur de cette méthode, à qui il arrive souvent de devoir développer en BASIC des programmes un peu copieux, s'y prend de la façon suivante.

J'écris mon programme sur une feuille blanche et en bon français : je voudrais qu'il se passe ceci, puis cela et enfin ceci. Toujours en français, je précise de plus en plus chacune des étapes. Au bout d'un moment, quelques mots de BASIC apparaissent dans ce discours. Enfin, certaines tâches particulières peuvent être entièrement programmées dans un sous-programme que je peux tester sur le MO6. Le sous-programme, c'est le bon outil. il faut donc commencer par parler de lui...

Exemples

Périmètre et surface

```
1 '
2 ' PERIMETRE ET SURFACE
3 '
20 LON=5.6 : LAR=2.8
30 GOSUB 1000
40 GOSUB 2000
999 END
1000 '
1001 ' PERIMETRE
1002 '
1010 PER=(LON+LAR)*2
1020 PRINT "PERIMETRE :" ; PER
1099 RETURN
2000 '
2002 ' SURFACE
2003 '
2010 SUR=LON*LAR
2020 PRINT "SURFACE :" ; SUR
2099 RETURN

RUN
PERIMETRE : 16.8
SURFACE : 15.68
```

Commentaires : il va sans dire qu'on pourrait faire l'économie des sous-programmes. On y gagnerait en concision mais on n'y prendrait pas de bonnes habitudes de programmation.

Conversions

L'utilité principale d'un sous-programme apparaît clairement lorsqu'il est appelé plusieurs fois. Aujourd'hui le dollar est à 6.90 francs.

```
1 '
2 ' CONVERSIONS FRANC DOLLAR
3 '
20 DF=6.90
30 SD=15 : GOSUB 1000
40 SD=3850.65 : GOSUB 1000
50 SF=60 : GOSUB 2000
60 SF=1000 : GOSUB 2000
999 END
1000 '
1001 ' DOLLAR-FRANC
1002 '
1010 SF=DF*SD
1020 PRINT SD ; " DOLLARS= " ; SF ; "FRANCS"
1099 RETURN
2000 '
2001 ' FRANC---DOLLAR
2002 '
2010 SD=INT((SF/DF*100)/100)
2020 PRINT SF ; "FRANCS=" ; SD ; "DOLLARS"
2099 RETURN
```

```
RUN
15 DOLLARS=6.90 FRANCS
3850.65 DOLLARS=6.90 FRANCS
60 FRANCS=6.90 DOLLARS
1000 FRANCS=6.90 DOLLARS
```

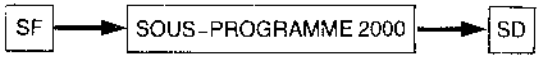
Commentaires : Variables :

— DF est le cours du dollar en francs
— SD est une somme en dollars
— SF est une somme en francs

Le sous-programme 1000 convertit les dollars en francs. Connaissant SD, il calcule SF suivant une formule évidente. On dit que SD est une variable d'entrée et SF une variable de sortie.

Le sous-programme est une machine qui effectue une certaine tâche. Il faut lui fournir la matière première (les variables d'entrée) pour qu'il donne les résultats (les variables de sortie).

Le sous-programme 2000 convertit les francs en dollars. Les variables d'entrée et de sortie sont permutées.



La conversion des francs en dollars se fait par l'intermédiaire d'une division. La formule de la ligne 2010 donne le résultat de cette division avec deux chiffres significatifs. Si SF/DF = 25.64387 alors 100*(SF/DF) = 2464.387, puis INT(100*(SF/DF)) = 2564 et enfin INT(100*(SF/DF))/100=25.64.

Arrondi

Voici une façon tout à fait stupéfiante de perfectionner la méthode précédente pour réaliser un vrai arrondi, c'est-à-dire que 8,348 soit arrondi à 8,35.

```
SD=INT((SF/DF)*100+0.5)/100
```

Le simple fait d'ajouter 0,5 fait basculer le nombre du bon côté. Pour les impatients, signalons qu'une instruction d'affichage spécialisée, PRINTUSING, réalise entre autre d'excellents arrondis (voir module 42).

Module 24 Trois dés

Projet

Faire afficher au hasard les faces de trois dés. Cette petite routine pourra être réutilisée dans un programme plus ambitieux (421).

Analyse

A. Tirer au hasard un nombre entier entre 1 et 6.
B. Suivant le nombre tiré, envoyer dans l'un des six sous-programmes qui afficheront la face.
C. Recommencer A et B trois fois.

Instructions BASIC mobilisées

Pour A on utilisera RND qui fournit des nombres aléatoires (module 17) et INT qui donne des valeurs entières. Pour B, il faut bien réfléchir pour un affichage économique. Seule l'instruction PRINT est nécessaire. Pour les points du dé, on utilisera l'astérisque (*), le signe – pour les bords horizontaux et ! pour les bords verticaux. Pour afficher la face 6, on pourrait enchaîner les instructions.

```
PRINT "-------"
PRINT "! * * !"
PRINT "! * * !"
PRINT "! * * !"
PRINT "-------"
```

C'est alors qu'on s'aperçoit que ce résultat pourrait être obtenu d'une façon plus économique puisque la même chaîne est imprimée plusieurs fois. En poussant le raisonnement un peu plus loin, on établit une liste de cinq chaînes élémentaires qui permettront d'afficher les six faces.

Les chaînes élémentaires :

```
"!   !";"! *   !";"!   *!";"! * *!";"! * *!";
```

Les six faces :

L'ordre et les chaînes choisies décideront de la face affichée. Il reste à voir comment brancher le programme sur l'un des six sous-programmes d'affichage.

ON... GOSUB

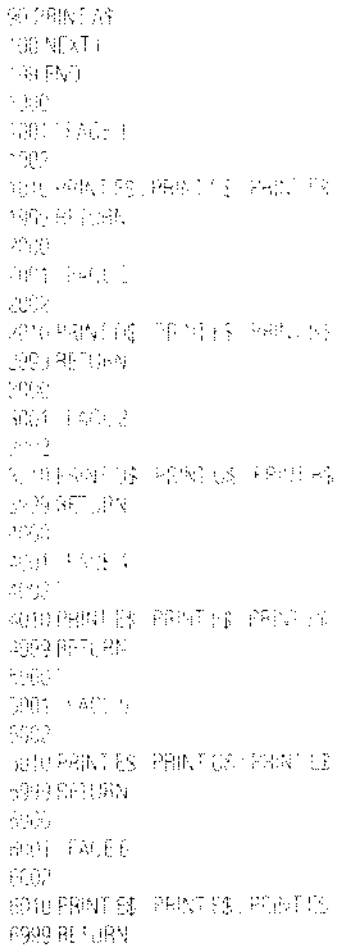
En une seule instruction, on obtient l'effet désiré. Si K est un nombre entier, l'instruction ON K GOSUB 1000, 2000, 3000 envoie au sous-programme 1000 si K vaut 1, au sous-programme 2000 si K vaut 2, au sous-programme 3000 si K vaut 3 et ignore cette instruction si K est supérieur à 3. Le nombre des valeurs possibles pour K, l'ordre et la numérotation des sous-programmes sont fixés par l'utilisateur.

Pour C on utilise une boucle FOR... NEXT.

Remarque : il nous manque toujours l'astuce pour obtenir un "véritable hasard" (voir module 29), la ligne 40 est destinée à recevoir cet ajout.

Programme

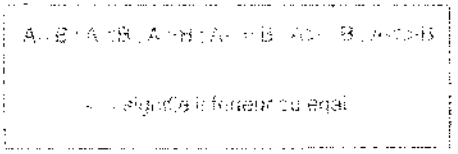
```
10 REM TROIS DES
20 A$="-------":B$="! * * !":C$="! * !"
30 D$="!   !":E$="! *   !":F$="!   * !"
40 REM EMPLACEMENT POUR RANDOMIZE
50 REM TIRAGES
60 FOR I=1 TO 3
70 K=1+INT(RND*6):PRINT A$
80 ON K GOSUB 1000,2000,3000,4000,5000,6000
```



Les jeux sont faits ! Rien ne va plus !
La ligne 40 est réservée pour le brassage des nombres aléatoires (voir le module 29).

Introduction au test

Posons un problème très simple : le micro-ordinateur est-il capable de distinguer un nombre (entier) pair d'un nombre impair ? Comment faisons-nous, nous-même ? On nous a appris à l'école qu'il suffit de regarder le chiffre des unités. Si c'est 0, 2, 4, 6 ou 8 alors le nombre est pair, sinon il est impair. D'une façon plus prosaïque, il suffit de diviser le nombre par 2 et de regarder si la division tombe juste ou pas. Toute la difficulté justement tient dans ce regard que l'ordinateur n'a pas. L'instruction de test va lui en donner un.
La syntaxe du test en BASIC utilise les mots anglais IF (si) et THEN (alors). Dans la plupart de nos exemples, IF sera suivi d'une relation de comparaison entre les contenus de deux variables de même type ou d'une variable et d'une constante de même type. En ce qui concerne les variables numériques, la relation de comparaison peut prendre six formes.



Attention !

Le signe = comme relation de comparaison entre les contenus des variables A et B, placé derrière IF, n'a rien à voir avec le signe = comme instruction d'affectation. En tant que test, il ne change pas le contenu des variables en cause.

Module 25 Test

Voici la structure complète d'un test en BASIC :

si [condition] alors [faire ceci] sinon [faire cela]

ce qui s'écrit

IF | condition | THEN | faire ceci | ELSE | faire cela |

Dans ce cadre, l'emploi de ELSE est facultatif alors que celui de THEN ne l'est pas. Mais si THEN et ELSE sont tous deux utilisés, une fois les instructions suivant ces deux mots exécutées, le programme se poursuit en séquence. Le meilleur usage que l'on puisse faire d'une telle séquence est d'aiguiller le programme sur deux sous-programmes différents en fonction du résultat du test. Par exemple,

```
IF A=B THEN GOSUB 1000 ELSE GOSUB 2000
```

qui revient au même que (et c'est intéressant de le remarquer) :

```
IF A<>B THEN GOSUB 2000 ELSE GOSUB 1000
```

Introduction aux opérateurs logiques

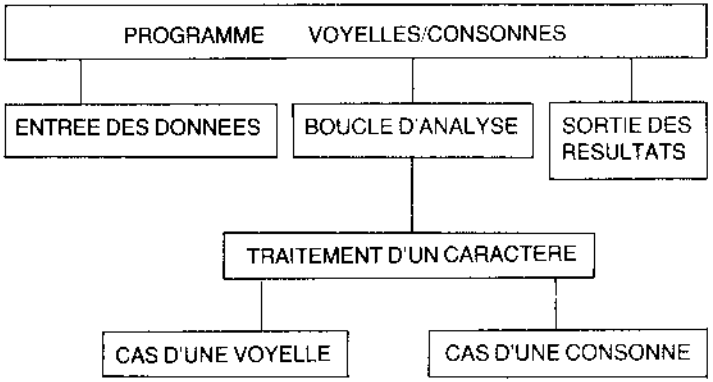
Il est possible de regrouper plusieurs conditions dans la même instruction de test IF... THEN... ELSE, au moyen des opérateurs logiques. Les opérateurs logiques utilisables sont AND (ET), OR (OU) et XOR (OU exclusif). Ce dernier opérateur signifie "l'un ou l'autre, mais pas les deux à la fois". Vous pouvez écrire l'opposé d'une condition avec NOT. Les opérateurs logiques seront étudiés en détail au module 35. Pour l'instant, l'intuition suffit à comprendre les exemples qui suivent.

Voyelles et consonnes

Voici un exemple qui fait le tri entre les voyelles et les consonnes d'une chaîne de caractères. Tous les caractères n'appartenant pas à l'une de ces catégories sont ignorés.

Analyse

1. Prendre chaque caractère de la chaîne, l'un après l'autre.
2. Tester s'il s'agit bien d'une lettre de l'alphabet.
3. Tester s'il s'agit d'une voyelle ou d'une consonne.
4. Ranger les voyelles et les consonnes dans deux chaînes de caractères réservées à cet effet.



Instructions BASIC mobilisées

1. On extrait les caractères grâce à MID$ (voir module 8).
2. On teste avec IF...THEN grâce au code ASCII (voir module 10) qui permet de repérer les lettres majuscules (ELSE facultatif).
3. Les voyelles et consonnes sont empilées par concaténation (module 15) dans deux chaînes (V$ et C$).

Le programme

```
1 '
2 ' VOYELLES-CONSONNES
20 A$="ALI-BABA ET LES 40 VOLEURS."
30 V$="" : C$="" : L=LEN(A$)
```

```
40 REM BOUCLE D'ANALYSE
50 FOR K=1 TO L
60 LA$=MID$(A$,K,1)
70 IF LA$ >="A" AND LA$ <="Z" THEN GOSUB 1000
80 NEXT K
90 REM BILAN
100 PRINT A$
110 PRINT "VOYELLES :";V$
120 PRINT "CONSONNES:";C$
999 END
1000 '
1001 ' TRAITEMENT D'UN CARACTERE
1002 '
1110 TLA=INSTR("AEIOUY",LA$)
1120 IF TLA=0 THEN GOSUB 12000 ELSE
     GOSUB 11000
1999 RETURN
11000 '
11001 ' VOYELLES
11002 '
11010 V$=V$+LA$
11999 RETURN
12000 '
12001 ' CONSONNES
12002 '
12010 C$=C$+LA$
12999 RETURN

RUN

ALI-BABA ET LES 40 VOLEURS.
VOYELLES : AIAAEEOEU
CONSONNES : LBBTLSVLRS
```

Commentaires :

70 : Premier test. Détermine si un caractère est une lettre majuscule ou non.

1110 : L'instruction INSTR permet de repérer la présence d'une chaîne dans une autre chaîne. Ici, on regarde si le caractère LA$ est présent dans la chaîne formée des voyelles ("AEIOUY"). Si c'est le cas, la variable TLA prend la valeur du rang de la voyelle repérée. Si TLA est nul, on peut donc conclure que la lettre est une consonne.
1120 : Les sous-programmes 11000 et 12000 concatènent le caractère dans l'une des deux chaînes, C$ ou V$.

L'instruction IF...THEN...ELSE ouvre de grandes perspectives. Remarquons que plusieurs instructions IF peuvent s'emboîter :

```
IF (condition 1) THEN (IF (condition 2)...)
                 ELSE (IF (condition 3)...)
```

Attention alors à ne pas s'embrouiller dans toutes ces conditions. Pour finir et pour montrer que l'on peut traiter des instructions dans un IF...THEN...ELSE plutôt que dans des sous-programmes, résolvons notre petit problème de parité.

Divisons juste.

Pour savoir si la division de A par 2 tombe juste, il faut diviser A par 2. Elle est juste s'il n'y a pas de partie décimale, c'est-à-dire si la partie entière du quotient est égale au quotient. C'est une façon bien curieuse de voir les choses mais si vous vous placez quelques secondes à la place du micro-ordinateur, vous constaterez que pour lui c'est tout naturel. Si A/2 = INT(A/2), alors la division tombe juste.

```
1 '
2 ' PARITE
3 '
20 X=2367
30 Q1=X/2 : Q2=INT(Q1)
40 IF Q1=Q2 THEN P$="PAIR" ELSE
   P$="IMPAIR"
60 PRINT X ; "EST " : P$
100 END

RUN
2367 EST IMPAIR
OK
```

Module 26 — Des boucles sans indice

Boucler, vous savez ! Cependant, l'instruction FOR...NEXT que l'on utilise déjà depuis quelques temps a un inconvénient : il faut connaître à l'avance le nombre d'itérations, c'est-à-dire le nombre de répétitions. Cette restriction peut être gênante dans certains cas. Nous allons voir que l'utilisation du couple DO...LOOP résout ce type de problème.
Le début de boucle est marqué par DO (faire, en anglais).
La fin de boucle est signalée par LOOP (boucler, en anglais).
Toutes les instructions comprises entre DO et LOOP seront répétées.
Pour éviter une répétition à l'infini, il est nécessaire de placer quelque part un test de fin de boucle. La forme de ce test est la suivante :
IF (condition de sortie) THEN EXIT

L'instruction EXIT provoque un branchement à l'instruction qui suit LOOP.
Exemple : Imaginons qu'on lance deux dés et qu'on s'arrête dès que les deux faces sont identiques. La succession des tirages aléatoires est faite dans une boucle DO...LOOP. On considère que l'on a gagné si on réussit à tirer une paire en moins de six tentatives.

Note : comme on utilise une fonction de hasard, on a toujours le même problème de "vrai hasard". Pour cela, voir le module 29.

```
1
2 ' PAIRE EN 6 COUPS
3 '
20 REM VRAI HASARD
30 COMPTEUR = 1
40 DO
50 F1 = 1 + INT(RND*6)
60 F2 = 1 + INT(RND*6)
70 IF F1 = F2 THEN EXIT
80 COMPTEUR = COMPTEUR + 1
90 LOOP
100 REM VERDICT
110 IF COMPTEUR <= 6 THEN PRINT "GAGNE"
    ELSE PRINT "PERDU"
120 PRINT "COMPTEUR" COMPTEUR
130 END
```

Sortie de boucle et drapeau

Ce petit programme est à comparer avec le suivant qui utilise l'instruction FOR...NEXT avec une longueur de boucle égale à six. Si la paire sort au premier tirage, on peut considérer qu'il est inutile de poursuivre, d'où l'intérêt de sortir de la boucle sans la dérouler entièrement afin de ne pas perdre de temps.

```
10 REM PAIRE EN 6 COUPS
20 FLAG = 0
30 FOR K = 1 TO 6
40 F1 = 1 + INT(RND*6)
50 F2 = 1 + INT(RND*6)
60 IF F1 = F2 THEN FLAG = 1 : EXIT
70 NEXT K
80 REM VERDICT
90 IF FLAG = 1 THEN PRINT "GAGNE"
   ELSE PRINT "PERDU"
100 END
```

Commentaires
20. FLAG signifie drapeau et nous avons choisi ce nom de variable car elle va précisément jouer le rôle d'un drapeau. Au départ, FLAG vaut 0, le drapeau est baissé.
50. Le test pour savoir si les deux font la paire. Si c'est oui, on fait deux choses très importantes. On lève le drapeau en mettant 1 dans FLAG. On provoque la sortie de boucle par l'instruction EXIT qui restitue la valeur de K au moment de la sortie.
90. Si le drapeau est levé, c'est qu'une paire est sortie.

En BASIC, un drapeau est une variable qui indique qu'une instruction a été exécutée soit par passage sur une ligne particulière, soit dans l'analyse d'un test.

Pour les amateurs de probabilités
Il est intéressant de calculer la probabilité de réussite offerte par le jeu précédent: probabilité de tirer une paire en moins de six tirages. Puisqu'il s'agit d'informatique, on peut écrire un programme de simulation qui fait 100 fois l'expérience et indique combien de fois une paire est sortie avant le sixième lancer. On en profitera pour mélanger les deux sortes de boucles.

```
1 '
2 ' SIMULATION
3 '
20 REM PLACE DU HASARD
30 NP=0 CC=1
40 DO
50 FOR K=1 TO 6
60 F1=1+INT(RND*6)
70 F2=1+INT(RND*6)
80 IF F1=F2 THEN NP=NP+1 : EXIT
90 NEXT K
100 IF CC=100 THEN EXIT
110 CC=CC+1
120 LOOP
130 PRINT NP
140 END
```

Les expériences faites semblent montrer un pourcentage de réussite voisin de 60%. Les distingués probabilistes pourront-ils donner une justification théorique à ce résultat expérimental?

Commentaires
Le drapeau a été supprimé puisque inutile. NP est un compteur relevant le nombre de paires. CC est le compteur du nombre d'essais.

L'instruction HEAD

La tortue fait des rebonds... Notre but est de faire parcourir la trajectoire suivante à la tortue.

```
1 '
2 ' REBOND
3 '
20 CLS
30 TURTLE 1,0,100
40 SHOW1,1
50 TRACE 1
60 DO
70 FWD5:HEAD4:ROT4
80 IF HEAD=68 THEN HEAD 124
90 LOOP
```

Commentaires
Pour arrêter la tortue lorsqu'elle est verticale, on utilise la fonction HEAD (même nom que l'instruction) qui donne la valeur de la direction de la tortue (de même que la fonction ZOOM donne la taille).
80. Lorsque la direction atteint 68 alors elle doit tourner de 124.
70. La tortue vient de tracer un segment vertical, sa direction est déjà 68 puisque l'augmentation par l'instruction HEAD se fait après le tracé par FWD. Ainsi le dernier segment tracé correspond à HEAD=68. Il suffit alors d'ajouter 124 à la direction de la tortue pour qu'elle se retrouve verticale dans l'autre sens.

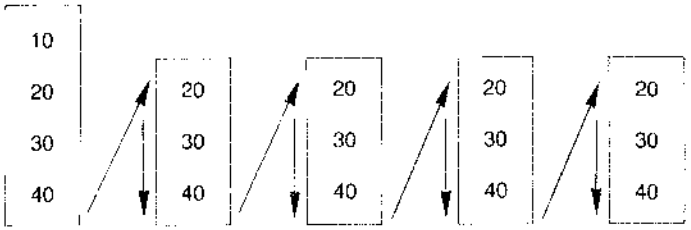
Module 27 Branchement automatique

Il est temps de présenter la star des instructions BASIC, le célèbre GOTO qui signifie ALLER A et qui permet, de même que GOSUB, d'aiguiller le déroulement du programme sur une autre série d'instructions. Mais à la différence de GOSUB, l'instruction GOTO n'assure pas le retour à l'envoyeur, ce qui est un désavantage certain pour GOTO.

Ceci dit, GOTO est une instruction très pratique qui plaît beaucoup au débutant car elle autorise des réalisations spectaculaires à peu de frais. Le spectacle dans notre premier exemple tient dans l'affichage de beaucoup de nombres à l'écran. De plus, ce programme ne s'arrête pas tout seul (inutile de placer l'instruction END), et nous conseillons au lecteur de consulter le module 19 avant de tenter cette expérience car le déroulement infernal de ce programme ne s'arrêtera que sur une intervention extérieure.

```
10 A=0
20 PRINT A;
30 A=A+1
40 GOTO 20
```

L'instruction GOTO 20 interrompt le déroulement séquentiel du programme en l'aiguillant sur la ligne 20. Après l'exécution de la ligne 20 et en l'absence d'une autre instruction de branchement, le déroulement séquentiel reprend jusqu'à la ligne 40 qui fait remonter en 20, etc.
Les lignes de programme dans l'ordre de l'exécution :



Le passage en ligne 20 imprime le contenu de la variable A, le passage en ligne 30 augmente d'une unité le contenu de cette variable. Vous ne serez donc pas surpris du résultat.

```
RUN
 0  1  2  3  4  5  6  7  8  9  10  11
 12  13  14  15  16  17  18  19  20  21
 22  23  24  25  26  27  28  29  30  31
 32  33  34  35  36  37  38  39  40  41
 42  43  44  45  46  47  48  49  50  51
 52  53  54  55  56  57  58  59  60  61
 62  63  64  65  66  67  68  69  70  71
```

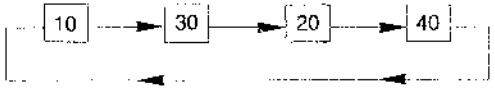
Le déroulement du programme a été interrompu par CNT-C. Sinon, nous y serions encore.
Puisque nous en sommes là, autant pousser l'instruction GOTO jusque dans ses derniers retranchements avec deux programmes absurdes qui ne font rien en y passant un temps infini.

```
10 GOTO 10
```

Un programme absurde qu'il faudra bien arrêter par CNT-C.

```
10 GOTO 30
20 GOTO 40
30 GOTO 20
40 GOTO 10
```

Pas beaucoup plus intelligent à vrai dire mais il est intéressant de suivre l'ordre des exécutions.



Autant GOSUB tend à structurer les programmes, autant GOTO a tendance à semer la confusion.

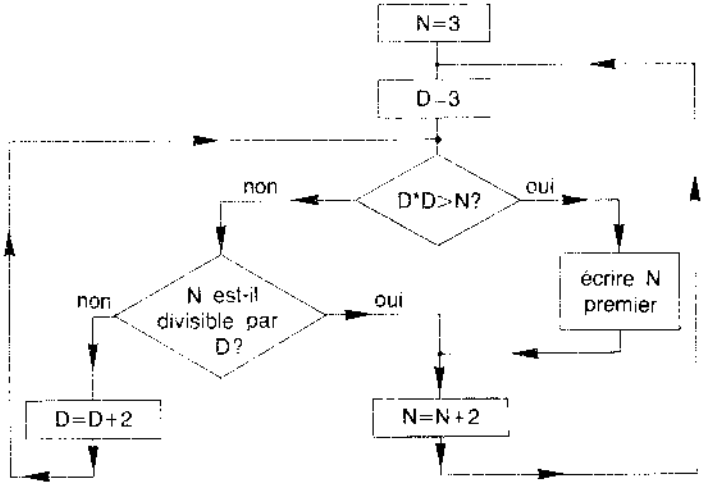
Exemple: les nombres premiers

Depuis la nuit des temps, les nombres premiers exercent sur les mathématiciens amateurs et professionnels une fascination qui ne se dément pas.
Rappelons qu'un nombre premier est un nombre entier divisible par aucun nombre que 1 et lui-même: 19 est premier, 20 ne l'est pas. On sait que les nombres premiers sont infiniment nombreux et qu'ils sont répartis parmi les autres d'une façon très mystérieuse et qui le restera toujours.

On peut écrire un programme très simple qui donne la liste des nombres premiers. La méthode utilisée peut être plus ou moins élégante. La pire de toutes consisterait à diviser le nombre examiné par tous ceux qui lui sont inférieurs. En réfléchissant un peu, on peut apporter les aménagements suivants:
1. On n'examinera que les nombres impairs à partir de 3 puisque manifestement les nombres pairs ne sont pas premiers.
2. Pour une raison analogue, on ne cherchera de diviseur que parmi les nombres impairs: un nombre impair ne peut pas être divisible par un nombre pair.
3. Plus subtil: pour savoir si N est premier, il suffit de chercher un diviseur éventuel inférieur à la racine carrée de N.

Organigramme

Un organigramme est un schéma qui résume l'algorithme d'un programme. On peut être pour, on peut être contre...
Nous ne sommes pas de chauds partisans des organigrammes qui sont trop souvent établis a posteriori. Voici donc l'organigramme du programme de recherche des nombres premiers qui permet tout de même de mieux comprendre les multiples branchements du programme.



Le programme suit les différents branchements grâce aux GOTO.

```
10 REM NOMBRES PREMIERS
20 N=3
30 D=3
40 IF D*D>N THEN GOTO 70
50 Q1 : N/D : Q2=INT(Q1)
60 IF Q1=Q2 THEN GOTO 80 ELSE D=D+2 : GOTO 40
70 PRINT N.
80 N=N+2 : GOTO 30
```

Remarque: dans l'instruction IF...THEN, le GOTO est facultatif; ainsi, on peut taper en 40: IF D*D>N THEN 70.

Module 28 Des entrées

ON...GOTO

Cette syntaxe fonctionne d'une façon similaire à ON...GOSUB (voir module 24) à la différence près que GOTO n'assure pas le retour. Cependant, elle permet de limiter les nombres d'instructions en cas de branchements multiples, en fonction des valeurs prises par une variable.

```
100 IF K=1 THEN GOTO 150
110 IF K=2 THEN GOTO 200
120 IF K=3 THEN GOTO 250
```

est équivalent à :

```
100 ON K GOTO 150,200,250
```

Entrées au clavier

Idéalement, un programme sur ordinateur est destiné à satisfaire un besoin: besoin de jeu, besoin de résultats numériques, besoin de créativité, besoin de distraction, etc. Jamais l'ordinateur ne pourra satisfaire tous les besoins des hommes, inutile de le préciser. Par exemple, le besoin d'affection.

```
10 PRINT "VOUS ETES BEAU  VOUS ETES GRAN
D. VOUS ETES FORT. JE VOUS AIME!"
20 GOTO 10
```

Ce programme a peu de chance de satisfaire celui à qui il est destiné; c'est un peu sommaire et trop impersonnel. L'instruction INPUT intervient alors pour améliorer le processus.

Personnalisation

L'instruction INPUT A introduite dans un programme a les conséquences suivantes :

1. Elle interrompt le déroulement du programme.
2. Elle met aussitôt le MO6 "à l'écoute" du clavier et affiche un point d'interrogation.
3. Toutes les touches frappées au clavier sont stockées dans une mémoire tampon.
4. La touche ENTREE envoie le contenu de la mémoire tampon dans la variable dont le nom suit obligatoirement le mot INPUT, ici, A, variable numérique.
5. Si le contenu de la mémoire tampon ne correspond pas au type de la variable, on obtient le message d'erreur REDO (refaire), la mémoire tampon se vide et le programme revient sous INPUT.
6. Si le type de la variable a été respecté, le programme continue en séquence avec le nouveau contenu de A.

Exemples

```
10 INPUT A
RUN
? 1515 [ENT]
OK
PRINT A
1515
OK
RUN
```

```
? ZUT [ENT]        <-- Le type de variable (numérique)
Redo                   ne correspond pas aux caractères
? 1848 [ENT]           tapés (chaîne de caractères)
OK
PRINT A
1848
OK
```

```
10 PRINT "QUEL EST VOTRE PRENOM :" ;
20 INPUT P$
30 PRINT "QUEL JOLI PRENOM," ; P$ ;"!"
40 END
RUN
QUEL EST VOTRE PRENOM :? RENE [ENT]
QUEL JOLI PRENOM, RENE!
OK
```

Aménagements

1. Les lignes 10 et 20 du programme précédent peuvent se résumer en une seule ligne.

```
10 INPUT "QUEL EST VOTRE PRENOM :" ; P$
```

2. On peut enregistrer plusieurs variables sous une seule instruction INPUT en les séparant par des virgules.

```
10 INPUT A,B,C
20 PRINT A+B+C
RUN
? 3,4,5
12
OK
```

```
10 INPUT "NOM, AGE :" ; N$,A
20 PRINT N$ ;" A " ; A ;" ANS."
RUN
NOM,AGE : ? RENE,34
RENE A 34 ANS
OK
```

Si vous devez saisir une seule réponse sous INPUT, attention à ne pas y inclure de virgule, vous auriez droit à un message désagréable du genre "Redo from start" (Recommencez au début).
C'est compréhensible: l'interpréteur BASIC considère que les réponses à l'instruction INPUT sont séparées par des virgules. Donc, s'il y a une virgule dans la réponse, il croit qu'il y a deux réponses et comme il n'en attendait qu'une...
Si vous tenez vraiment à la virgule, vous pourrez la garder mais il faudra mettre votre réponse entre deux guillemets. Ce peut être facheux pour la saisie de textes.
Si vous souhaitez que la réponse puisse contenir vraiment n'importe quoi, il faut utiliser l'instruction LINE INPUT.

```
LINE INPUT "Quel mot";A$
```

permet d'attribuer à la chaîne A$ n'importe quelle suite de caractères.

Remarque et conclusion:
L'instruction LINE INPUT n'affiche pas de point d'interrogation et ignore les virgules.

Module 29 Prise au vol

L'instruction INPUT (module 28) est réservée à un usage tranquille de l'interactivité. En passant sur INPUT, le programme s'arrête et attend gentiment que l'utilisateur tape sur la touche ENTREE. Cette procédure est évidemment incompatible avec un jeu rapide ou une analyse fine des caractères frappés. Il existe une instruction BASIC courante qui permet de tester l'état du clavier et éventuellement de poursuivre le programme : INKEY$.
Lorsque le programme rencontre l'instruction INKEY$, il ouvre une mémoire tampon susceptible de recevoir *un seul* caractère, et y place le caractère tapé au clavier. Ce caractère peut être placé dans une variable chaîne de caractères par une affectation classique (A$ = INKEY$) de façon à être analysé facilement. Les différences essentielles entre INPUT et INKEY$ sont les suivantes :

1. INKEY$ n'enregistre qu'un caractère.
2. Ce caractère est de type chaîne.
3. L'appui de ENTREE est inutile avec INKEY$.
4. INKEY$ n'arrête pas le programme.
5. Le caractère frappé sous INKEY$ n'est pas affiché.

En général, l'instruction INKEY$ est aussitôt suivie de tests qui permettront de brancher le programme en fonction des touches frappées. On commence souvent par regarder s'il y a quelque chose dans INKEY$.

```
10 PRINT "VITE, APPUYEZ SUR UNE TOUCHE!"
20 IF INKEY$="" THEN GOTO 10
30 PRINT "OUF...MERCI!"
40 END
```

Sans commentaire.
Plus subtil maintenant.

```
10 PRINT "VITE, APPUYEZ SUR A!"
20 A$=INKEY$
30 IF A$<>"A" THEN GOTO 10
40 PRINT "OUF... MERCI!"
50 END
```

Comme on est sûr que A$ contient une chaîne de longueur maximum 1, on peut donc faire porter le test sur le code ASCII de cet unique caractère. C'est indispensable pour tester l'appui sur les touches de déplacement du curseur ou la touche ENTREE.

OUI ou NON ?
Voilà bien une alternative souvent proposée à l'utilisateur d'un programme. Si une question posée appelle une réponse entre Oui et Non, on testera l'appui d'une seule touche pour assouplir l'interactivité : dans une version française, on renvoie sur INKEY$ dès que la touche appuyée est différente de O et N. Cette précaution interdit à l'utilisateur de pouvoir répondre des grossièretés...

```
10 PRINT "ALLEZ-VOUS BIEN?"
20 DO
30 A$=INKEY$
40 IF A$="O" THEN GOSUB 100:EXIT
50 IF A$="N" THEN GOSUB 200:EXIT
60 LOOP
99 END
100 '
101 ' OUI
102 '
110 PRINT "TANT MIEUX!"
199 RETURN
200 '
201 ' NON
202 '
210 PRINT "TANT PIS!"
299 RETURN
```

Le vrai hasard

L'instruction INKEY$ va nous permettre d'obtenir des tirages au hasard véritablement hasardeux.

```
10 PRINT,"TAPEZ UNE TOUCHE"
20 X=RND : IF INKEY$="" THEN GOTO 20
30 PRINT RND
40 END
```

A la ligne 20, on boucle très vite tant qu'aucune touche n'est frappée. La variable X est à chaque fois chargée d'un nombre aléatoire. Le temps d'attente étant nécessairement variable à chaque usage, on obtient à chaque déroulement une initialisation différente de la suite utilisée.
A partir de ce programme, faites RUN plusieurs fois et vous verrez bien.

Tester plusieurs caractères au clavier: INPUT$

Dans un dialogue avec un programme, il peut être intéressant d'avoir un contrôle automatique sur le nombre de caractères entrés par l'utilisateur indépendamment de ce qu'il a réellement tapé. L'instruction INPUT$ est prévue pour cette circonstance.
Attention, il ne s'agit pas d'une instruction comme INPUT mais d'une fonction comme RIGHT$; elle rend donc un résultat qu'il faut affecter à une variable, ou traiter directement. On indique dans INPUT$ le nombre de caractères attendus, et la fonction retourne les caractères tapés au clavier. Le test O ou N peut alors se réécrire en utilisant INPUT$(1). Mais ce n'est pas ce cas qui nous intéresse le plus.
La possibilité de tester une réponse de plusieurs caractères peut servir à mettre des mots de passe à l'entrée de vos logiciels.

Exemple:

```
10 REM DEMONSTRATION
20 PRINT "MOT DE PASSE"
30 REP$=INPUT$(4)
40 IF REP$<>"LULU" THEN NEW
ELSE PRINT "BIENVENUE"
50 REM DEBUT DU PROGRAMME
999 END
```

Commentaires:

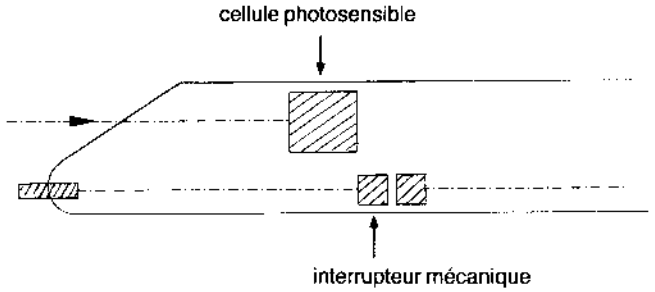
En ligne 30, l'interpréteur ne passe à la suite que si l'utilisateur a tapé quatre caractères. Pour tester la différence de deux chaînes, on utilise le signe <>, comme pour les nombres.
En ligne 40, l'instruction NEW supprime tout ce qui est présent en mémoire et vous êtes bien puni!

Remarque:

On objectera rapidement que cette méthode n'est pas très sûre. Il suffit en effet à l'utilisateur de taper LIST pour que les premières lignes du programme apparaissent à l'écran et que le mot de passe s'affiche en clair devant les yeux du pirate!

Module 30 Le crayon optique

Le crayon optique est un outil d'interactivité très puissant. De nombreux logiciels l'utilisent. Vous allez pouvoir en faire autant. Les commandes gouvernant le crayon optique sont très simples. L'état du crayon est déterminé par deux paramètres: l'état d'un interrupteur mécanique et l'état d'une cellule photosensible.



Il existe donc fort logiquement trois instructions qui renseignent complètement sur l'état du crayon.

1. *L'interrupteur seul.* PTRIG vaut −1 si l'interrupteur est enfoncé. PTRIG vaut 0 si l'interrupteur est relâché.

2. *La cellule seule.* INPEN X, Y charge un numéro de colonne graphique dans X et un numéro de ligne dans Y. Le crayon doit être bien orienté et à moins de 10 cm de l'écran. Le rouge et le noir ne sont pas perçus. Lorsque la cellule n'est pas activée, elle charge −1 dans X et dans Y.

3. *L'interrupteur et la cellule.* INPUTPEN X, Y fonctionne comme INPEN après fermeture de l'interrupteur, en général par appui sur l'écran.

Programme
Il affiche les coordonnées d'un point visé au crayon optique.

```
10 'POINT VISE
20 CLS
30 DO
40 INPEN C,L
50 LOCATE 0,0:PRINT "COLONNE";C,"LIGNE";L
60 LOOP
```

Commentaires
Dès que vous avez lancé l'exécution du programme, vous voyez s'afficher les coordonnées −1 et −1. Puis, en approchant le crayon optique, les nombres défilent rapidement. Vérifiez que le numéro de colonne varie bien de 0 à gauche à 319 à droite (il n'est même pas très facile de viser les points du bord!) et que le numéro de ligne varie de 0 en haut à 199 en bas.
Si les coordonnées s'obstinent à conserver les valeurs −1,
— ou bien vous visez obstinément en dehors de la zone utile,
— ou bien la couleur de fond est trop foncée (noir ou rouge),
— ou bien votre téléviseur n'envoie pas assez de lumière au crayon optique. Il faut dans ce cas augmenter la luminosité du téléviseur.

Variations
On peut visualiser le point visé en ajoutant une instruction PSET au bon endroit. L'écran se remplit alors de petits points correspondant à chaque endroit que vous avez pointé.
Pour afficher uniquement des points de coordonnées déterminées, et non pas les points visés, on peut utiliser l'instruction PTRIG qui fonctionne avec l'interrupteur du crayon optique.
Par exemple, l'instruction IF PTRIG THEN PSET (C,L) se lit: si le crayon optique est appuyé alors afficher le point C,L. Pour mieux comprendre le fonctionnement de ce test, rendez-vous au module 35.

Applications

Les applications du crayon optique sont innombrables dès l'instant où il permet à l'utilisateur de s'affranchir des contraintes du clavier: choix dans un menu, désignation précise d'un endroit de l'écran

sans avoir à déplacer le curseur et sans abîmer l'image par des repères disgracieux.
Que l'on songe simplement à la façon de choisir une case dans un quadrillage. Avec le seul clavier, il faut passer par un système de coordonnées et l'on imagine les problèmes de saisie que cela implique.
Le plus spectaculaire réside cependant dans la possibilité de dessiner sur l'écran. Le lecteur un tant soit peu familiarisé avec le maniement du BASIC admettra que les trois instructions précédemment décrites rendent la mise en œuvre de ce projet extrêmement simple.
Pour le tracé du dessin, on utilise PSET (X,Y) les instructions de la tortue.

```
5 CLS
10 TURTLE 0
20 DO
30 DO
40 INPEN X,Y
50 IF X<>-1 THEN EXIT
60 SHOW 0
70 LOOP
80 TRACE PTRIG : TURTLE 0,X,Y : SHOW 1
90 LOOP
```

Commentaires
40. Enregistrement d'une position.
80. Suivant que l'interrupteur est enfoncé ou pas, la tortue marque ou non son déplacement.

Voici un autre programme, tout à fait étonnant qui simule un élastique. Tapez le programme puis lancez-le (RUN). Commencez par pointer n'importe où sur l'écran. Levez ensuite le crayon et vous verrez un trait élastique suivre votre geste.
Ce programme utilise l'instruction INPEN qui fonctionne comme INPUTPEN mais sans que le crayon soit appuyé sur l'écran. Vous pouvez éloigner le bout du crayon jusqu'à 10 cm de l'écran. Au-delà, la luminosité est trop faible.

```
10 SCREEN 0,7,7 : CLS
20 X0=160 : Y0=100 : X1=160 : Y1=100
30 GOSUB 1000
40 X0=X2 : Y0=Y2 : GOTO 30
999 END
1000 '
1001 ' Déplacement élastique
1002 '
1010 INPEN X2,Y2 : IF X2<0 OR Y2<0 THEN 1010
1020 LINE (X0,Y0)-(X1,Y1),-8
1030 LINE (X0,Y0)-(X2,Y2),0
1040 X1=X2 : Y1=Y2
1050 IF PTRIG=0 THEN 1010
1999 RETURN
```

Commentaires
Le programme utilise trois paires de coordonnées :
— X0 et Y0 : extrémité fixe de l'élastique
— X1 et Y1 : ancienne extrémité mobile
— X2 et Y2 : nouvelle extrémité mobile
Le sous-programme 1000 teste la position du crayon (1010). Il efface l'ancien élastique (1020) dans la couleur de fond (-8). Il trace le nouvel élastique (1030) en noir. Il remplace l'ancien par le nouveau (1040). Lorsque l'interrupteur est enfoncé (1050), il sort du sous-programme. La ligne 40 du programme principal redéfinit les coordonnées de la nouvelle extrémité fixe de l'élastique.

Pour les heureux propriétaires d'une souris.
Il existe trois instructions de programmation de la souris calquées sur celles du crayon et dans lesquelles PEN est remplacé par MOUSE (souris anglaise).
INPEN → INMOUSE
INPUTPEN → INPUTMOUSE
PTRIG → MTRIG

Module 31 Dimension

Les tableaux permettent de ranger des données de même type destinées à un même usage sous un nom de variable commun comportant une numérotation entière qui permet de retrouver chaque donnée.

Dimension 1

Les tableaux à une dimension sont les plus simples: un nom de variable derrière lequel figure un indice placé entre parenthèses. La dimension du tableau est l'indice maximum. Dans un tableau A de type numérique de dimension 10, on peut enregistrer dix nombres appelés respectivement A(1), A(2), A(3), jusqu'à A(10).
Une contrainte très importante pour le programmeur est de prévoir dès le début la dimension de son tableau et de la déclarer par une instruction de type DIM A(10).
Déclarer dès le départ des dimensions très importantes pour être tranquille est une maladresse qu'il ne faut pas commettre car l'instruction DIM réserve de la place en mémoire. Une prévision trop gourmande risque d'être regrettée par la suite. Une autre contrainte importante est la suivante:

> Il est interdit de dimensionner plusieurs fois le même tableau

On peut définir des tableaux de nombres ou des tableaux de chaînes de caractères.

Note: Il est possible d'utiliser des tableaux sans les déclarer. Dans ce cas, la valeur maximale des indices du tableau est 10. Cela signifie qu'un tableau non dimensionné par DIM se voit réserver onze places par défaut (de 0 à 10). Néanmoins, pour la clarté des programmes, il est conseillé de dimensionner tous les tableaux utilisés.
Voici un programme qui construit un tableau de nombres entiers au hasard entre 0 et 100, et un tableau de lettres au hasard entre A et Z. Pour s'assurer que le travail a bien été fait, on fait imprimer deux éléments des tableaux.

```
10 REM TABLEAUX ALEATOIRES
20 DIM A(10),B$(10)
30 FOR I=1 TO 10
40 A(I)=INT(RND*100)
50 B$(I)=CHR$(65+INT(RND*26))
60 NEXT I
RUN
OK
PRINT A(4),B$(7)
4          C
```

Remarque: un tableau de chaîne a un nom se terminant par $ (sa taille doit être déclarée).

Dimensions quelconques

Les tableaux peuvent avoir un nombre quelconque de dimensions, mais en pratique, seuls les tableaux de dimension 1 et 2 sont utilisés. Les derniers sont intéressants pour les jeux sur quadrillage, très pratiqués en micro-informatique à cause des deux dimensions de l'écran.
Les amateurs de mathématiques penseront à juste titre qu'un tableau de dimension 2 est une matrice. Nous leur offrons donc un court programme qui calcule le produit de deux matrices dont les éléments sont des entiers calculés au hasard. En utilisant les instructions d'entrée (INPUT), ils n'auront pas de mal à en faire un vrai programme utilitaire (voir module 28).

Programme

```
10 REM PRODUIT DE MATRICES
20 N=3 : P=4 : Q=5
30 DIM A(N,P), B(P,Q), C(N,Q)
40 PRINT "MATRICE A"
50 FOR I=1 TO N : FOR J=1 TO P
60 A(I,J)=5-INT(RND*10) : PRINT A(I,J) ;
70 NEXT J : PRINT : NEXT I
```

```
80 PRINT "MATRICE B"
90 FOR I=1 TO P : FOR J=1 TO Q
100 B(I,J)=5-INT(RND*10) : PRINT B(I,J) ;
110 NEXT J : PRINT : NEXT I
120 REM PRODUIT
130 PRINT "MATRICE AXB"
140 FOR I=1 TO N : FOR J=1 TO Q
150 FOR K=1 TO P
160 C(I,J)=C(I,J)+A(I,K)*B(K,J)
170 NEXT K : PRINT C(I,J) ;
180 NEXT J : PRINT : NEXT I
200 END
```

```
MATRICE A          MATRICE B              MATRICE AXB
 0  0  5  1        4  5 -4  5  3        13  12   0 -15  25
 4 -1 -4  1       -4  1  3  3 -3         6   4 -19  38  -5
 5  5  3  0        3  3  0 -4  5         9  39  -5  28  15
                  -2 -3  0  5  0
```

Commentaires :

Nous laissons les fervents de mathématiques analyser ce programme en détail. Les boucles imbriquées 50-70 définissent et impriment la matrice A. Les boucles imbriquées 90-100 définissent et impriment la matrice B.
Entre les lignes 140 et 180, on ne compte pas moins de trois boucles imbriquées. Celles en I et J construisent la matrice produit. La boucle en K contient le calcul bien connu des termes du produit. La formule est en ligne 160.

Les tableaux sont beaucoup utilisés en informatique. En effet, ils permettent de manipuler sous un seul nom une quantité importante de variables.
On remarquera à cette occasion que toutes les variables stockées dans un tableau sont de même type. Il est impossible de mélanger des variables simple précision avec des variables double précision, encore moins avec des variables chaînes de caractères. Cela tient à la méthode de rangement utilisée à l'intérieur de la machine : elle exige que tous les éléments du tableau occupent le même nombre d'octets dans la mémoire car ainsi elle retrouve facilement ses petits !

La particularité d'un tableau tient aussi à la méthode d'accès aux éléments du tableau. Pour le MO6, aucune différence n'est faite entre le premier élément et le vingt-huitième élément du tableau. Il met aussi peu de temps à retrouver l'un que l'autre.
Cette remarque prendra toute son importance quand nous décrirons la méthode d'accès aux informations contenues dans un fichier séquentiel (voir modules 46-47). L'inconvénient majeur des tableaux fait le pendant de leur avantage. Ils résident en mémoire centrale — c'est pour cela qu'ils sont aisément manipulables — et ils occupent donc beaucoup de place mémoire. Stocker un grand nombre d'informations dans un tableau peut ainsi vous conduire à un blocage du programme par manque de place.

Nous avons ici mis le doigt sur un des problèmes fondamentaux de l'informatique : doit-on privilégier le temps d'accès aux informations (et ainsi le temps de traitement global) ou la place mémoire occupée. La plupart du temps, on ne peut pas faire les deux simultanément... cela serait trop facile !

Module 32 Données

Il existe trois façons différentes d'enregistrer le nombre 1515 sous le nom de variable MARIGNAN. Deux d'entre elles ont été présentées dans des modules précédents. La troisième (DATA-READ) est introduite ici.

```
10 MARIGNAN=1515
20 PRINT MARIGNAN
RUN
1515
OK
```
1

```
10 INPUT MARIGNAN
20 PRINT MARIGNAN
RUN
?1515
1515
OK
```
2

```
10 DATA 1515
20 READ MARIGNAN
30 PRINT MARIGNAN
RUN
1515
OK
```
3

Commentaires :
— La méthode 1 est la plus courte et la plus naturelle.
— La méthode 2 fait toute confiance à l'utilisateur, ce qui n'est pas la meilleure des formules.
— La méthode 3 utilise deux nouvelles instructions DATA et READ dont nous allons détailler le fonctionnement.

Lorsque le MO6 (plus exactement, l'interpréteur BASIC) rencontre une ligne DATA, il feint de l'ignorer comme une ligne REM. Le ou les nombres ou chaînes suivant le DATA sont parfaitement ignorés dans un premier temps.
Lorsque l'interpréteur rencontre une instruction READ suivie obligatoirement d'un nom de variable, il consulte le pointeur de données, lit la donnée correspondante et la range sous le nom de variable indiqué. Puis il incrémente son compteur de données en prévision du prochain READ. Les lignes DATA peuvent être placées à n'importe quel endroit dans le programme et comporter un nombre variable de données, de tous types, séparées par une virgule.
Ce type de protocole est une commodité donnée à l'utilisateur pour affecter un grand nombre de données à un grand nombre de variables. Voici par exemple une bonne façon d'enregistrer un tableau de nombres et de l'imprimer. On remarquera que la répartition des DATA dans le programme permet de reproduire le tableau tel qu'il est, et facilite donc la correction des erreurs.

```
10 DIM A(5,3)
20 DATA 1,8,5,3,2
30 DATA 4,6,7,2,9
40 DATA 1,0,6,6,3
50 FOR J=1 TO 3 : FOR I=1 TO 5
60 READ A(I,J) : PRINT A(I,J) ;
70 NEXT I : PRINT : NEXT J
RUN
1  8  5  3  2
4  6  7  2  9
1  0  6  6  3
```

RESTORE

Cette instruction redonne au pointeur de données sa valeur initiale afin que le premier READ puisse lire la première donnée derrière le premier DATA. Elle est utilisée lorsqu'on désire lire les données plusieurs fois dans le programme.

Types de variable

Une même ligne de DATA peut présenter des variables d'un type différent à condition que l'ordre de lecture tienne compte de ces différences.

```
10 DATA MARIGNAN, 1515
20 READ B$,D
```

Fin de lecture

Lorsque le nombre de données en DATA est susceptible de varier, on peut placer en queue de données une donnée supplémentaire fictive qui sera un signe de fin de lecture.
Voici un programme qui calcule une moyenne sur un nombre quelconque de notes placées en DATA, ces notes étant comprises entre 0 et 20. Le signe de fin de lecture est le nombre 99.

```
10 REM MOYENNE
20 DATA 11,14,13,8,10,5,999
30 N=0 : T=0 : RESTORE
35 DO
40 READ A
50 IF A>20 THEN EXIT
60 T=T+A : N=N+1
70 LOOP
80 PRINT "MOYENNE :" ; T/N
100 END
```

Exemple d'utilisation:
1. Lire trente et une valeurs représentant les différentes températures relevées à Paris au mois de juillet.
2. Calculer le minimum et le maximum des températures du mois.
3. Tracer la courbe d'évolution des températures (échelle = 5 lignes pour 1°).

On utilisera les fonctions mathématiques MIN et MAX pour calculer les minimum et maximum du point 2.
MIN (A,B) = plus petit des deux nombres A et B
MAX (A,B) = plus grand des deux nombres A et B

Programme

```
10 ' METEO
20 DIM TEMP(31)
30 CLS
40 FOR I=1 TO 31
50 READ TEMP(I)
60 NEXT I
99 DATA 25,27,26,30,29,33,28,32,31,33,29,28,31,30,26,29,
32,30,28,25,28,26,31,30,27,29,28,29,27,26,29
100 ' MINI-MAXI
110 TMIN=100 : TMAX=-50
120 FOR I=1 TO 31
130 TMIN=MIN(TEMP(I),TMIN)
140 TMAX=MAX(TEMP(I),TMAX)
150 NEXT I
160 PRINT "MAXIMUM" ; TMAX
170 PRINT "MINIMUM" ; TMIN
200 ' COURBE
210 PSET(0,200-TEMP(I)*5)
220 FOR I=2 TO 31
230 LINE-((I-1)*10,200-TEMP(I)*5)
240 NEXT I
999 END
```

Commentaires:
— L'ordre DATA est utilisé pour initialiser un tableau: TEMP.
— Le minimum est noté TMIN et le maximum TMAX; rappelons que les variables commencant par MIN ou MAX sont interdites puisque MIN et MAX sont des mots clés.
— 110. On initialise ces deux variables par des valeurs improbables (100 et −50).
— 130 et 140. On boucle sur les nombres traités et on garde à chaque fois le plus petit et le plus grand.

Module 33 Tri numérique

Les problèmes de tri constituent l'un des casse-têtes les plus sérieux offerts à la sagacité des informaticiens. Ne serait-ce que pour classer des nombres, du plus petit au plus grand, on pourrait y passer sa vie. S'il s'agit de petits échantillons, le problème n'est pas d'un grand intérêt car quelle que soit la méthode utilisée, ça ira vite! Par contre, s'il s'agit de plusieurs milliers de nombre, on a tout intérêt à minimiser le nombre des manipulations.
En deux pages, nous ne pouvons qu'évoquer le problème. Voici deux méthodes élémentaires pour trier agréablement de petits échantillons de nombres.

Méthodes du minimum

Méthode très bête: on regarde tous les nombres, on prend le plus petit, on recommence avec ceux qui restent jusqu'à ce qu'il n'y en ait plus.
Le programme trie N nombres tirés au hasard entre 0 et 100. On a fixé N à 10 mais rien n'interdit de changer.

```
10 REM TRI NUMERIQUE
20 N=10 : DIM A(N)
30 FOR I=1 TO N
40 A(I)=INT(RND*100) : PRINT A(I) ;
50 NEXT I : PRINT
60 REM C'EST PARTI !
70 FOR D=1 TO N-1
80 X=D : INF=A(D)
90 FOR K=D+1 TO N
100 IF A(K)<INF THEN X=K : INF=A(K)
110 NEXT K
120 SWAP A(X),A(D)
130 NEXT D
140 REM AFFICHAGE SUITE TRIEE
150 FOR I=1 TO N : PRINT A(I) ; : NEXT I
200 END
```

```
RUN
59  51  1  42  16  62  90  43  4  9
1  4  9  16  42  43  51  59  62  90
```

Commentaires

Deux boucles sont imbriquées. Dans la boucle intérieure, on cherche au-delà de A(D) le nombre le plus petit (INF). On note son indice (X). A la sortie de cette boucle, on permute A(D) et A(K) (ligne 120). Puis on augmente D de 1. Le tri est assez long. On s'en rend compte en faisant N=100.

Tri à bulle

C'est une façon très différente de procéder. Elle est en tout cas beaucoup moins naturelle. Ce qui ne l'empêche pas d'être tout autant efficace.
Son principe est simple. On regarde les nombres dans l'ordre et deux par deux. Si ces deux-là ne sont pas dans le bon ordre, on les permute et on passe aux suivants. Tant qu'il faut faire de telles permutations, on continue. Lorsque ce n'est plus nécessaire, c'est que le tri est terminé.
Ce programme utilise un drapeau (la bulle) qui remonte lentement vers le sommet de la suite.

```
10 REM TRI A BULLE
20 N=10 : DIM A(N)
30 FOR I=1 TO N
40 A(I)=INT(RND*100) : PRINT A(I) ;
50 NEXT I
60 REM C'EST PARTI !
70 DRA=0 : FOR I=1 TO N-1
80 IF A(I)<=A(I+1) THEN GOTO 100
90 SWAP A(I),A(I+1) : DRA=1
100 NEXT I
110 IF DRA=1 THEN GOTO 70
```

```
120 REM C'EST FINI
130 PRINT : FOR I=1 TO N
140 PRINT A(I) ; : NEXT I

RUN
66  40  59  94  27  65  3  93  38  57
3  27  38  40  57  59  65  66  93  94
```

Commentaires :
En 70, le drapeau (DRA) est baissé, c'est-à-dire mis à 0. On lit chaque nombre de la suite que l'on compare avec son suivant. S'ils sont dans le bon ordre (80), on continue. Sinon, on les permute (ligne 90). On lève le drapeau ; DRA=1.
Si une permutation au moins a été effectuée, le drapeau est levé et il faut recommencer. Lorsque toute la boucle est déroulée sans que le drapeau soit levé, c'est que le tri est terminé. On le vérifie en imprimant la suite ainsi arrangée.

Tri alphabétique

On pourrait croire que l'ordre alphabétique, et plus précisément l'ordre lexicographique, pose un problème délicat aux ordinateurs. Ce serait sans compter sur le fameux code ASCII qui considère chaque lettre majuscule de l'alphabet comme un nombre entier compris entre 65 et 90. Comme les nombres, les chaînes peuvent être comparées à l'aide des relations <, > ou =.
Dire qu'une chaîne AS est inférieure à une chaîne B$, c'est dire que A$ est avant B$ dans l'ordre ASCII.
Exemple :
"ab" < "bc"
"rouge" < "vert"
"abc" < "bc"

Les méthodes de tri examinées plus haut peuvent donc être simplement transposées à des tableaux de chaînes de caractères. Il suffit de mettre le signe $ derrière les noms de variables et le tour est joué. Bien entendu, la définition du tableau ne peut être faite aussi simplement que pour les nombres. Chaque élément devra être précisé explicitement au début du programme.
L'une des difficultés vient de la présence éventuelle de lettres minuscules codées entre 97 et 122. De ce simple point de vue, on a évidemment "B" < "a" ce qui est tout à fait fâcheux. Les relations de comparaison entre chaînes de caractères sont fondées sur le code ASCII sans qu'il soit possible d'y changer quoi que ce soit. Pour bien tenir compte des lettres minuscules, il faut écrire une petite routine qui transforme, dans une chaîne, les minuscules en majuscules. C'est une solution un peu brutale. On pourrait procéder autrement.

```
10 A$="Ali-Baba" : LA=LEN(A$) : AA$=""
20 FOR K=1 TO LA
30 CL=ASC(MID$(A$,K,1))
40 IF CL>96 AND CL<123 THEN CL=CL-32
50 AA$=AA$+CHR$(CL)
60 NEXT K
70 PRINT A$ : PRINT AA$
80 END
```

Commentaires :
10. AA$ est une chaîne vide dans laquelle on va recopier A$, caractère par caractère, en changeant les minuscules en majuscules.
30. CL est le code ASCII du K-ième caractère de A$.
40. Si CL est un code de lettre minuscule, on lui enlève 32 pour le ramener à la même place dans les lettres majuscules : le code de "A" est 65, celui de "a" est 97, et 97−65 = 32.
50. On empile les caractères dans AA$.
70. On remarque à l'impression que seules les minuscules ont été transformées en majuscules.

Module 34 — Vive les maths !

Fonctions mathématiques

Tous nos lecteurs ne sont sans doute pas mathématiciens. Pour ceux qui le sont ou qui s'intéressent de près ou de loin aux mathématiques, le tableau suivant donne la liste des fonctions mathématiques ainsi que leurs conditions d'emploi en BASIC.

Fonction	Symbole	Exemple	BASIC
Elévation à la puissance	↑	$y=x^2$	Y=X↑2
Racine carrée	SQR	$y=\sqrt{x}$	Y=SQR(X)
Logarithme népérien	LOG	y=Log(x)	Y=LOG(X)
Exponentielle	EXP	$y=e^x$	Y=EXP(X)
Sinus (radians)	SIN	y=sinx	Y=SIN(X)
Cosinus (radians)	COS	y=cosx	Y=COS(X)
Tangente (radians)	TAN	y=tgx	Y=TAN(X)
Arc tangente (résultat en radians)	ATN	x=Arc tg(y)	X=AT N(Y)
Partie entière	INT	n=E(x)	N=INT(X)
Valeur absolue	ABS	y=|x|	Y=ABS(X)
Valeur du signe	SGN	n=sign(x)	N=SGN(X)
Troncate le décimal ou entier	FIX	x_1=tronc(x)	X1=FIX(X)
Minimum	MIN	z=min(x,y)	Z=MIN(X,Y)
Maximum	MAX	z=max(x,y)	Z=MAX(X,Y

Exemple d'application

L'opération de division / est effectuée en poussant éventuellement après la virgule. Pour effectuer une division entière (division euclidienne), on peut faire appel à l'opération MOD (module) qui fournit le reste de la division entière. Exemple :

```
PRINT 256 MOD 7
 4
OK
```

256	7
46	36
4	

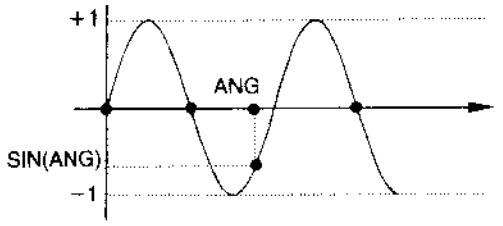
Barbouille

```
10 CLS : SCREEN 3,0,0
20 FOR X=0 TO 319
30 C=X MOD 16
40 LINE (0,0)-(X,199),C
50 NEXT X
60 END
```

320 lignes sont dessinées dans l'une des seize couleurs. Ce n'est pas long et c'est très beau. Le phénomène de bavure est expliqué dans le module 36.

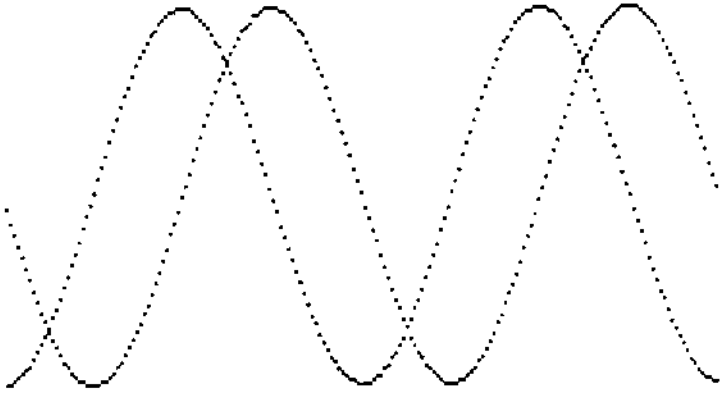
Sinusoïde

Si la variable ANG représente un angle exprimé en radians, SIN(ANG) est un nombre variant de façon sinusoïdale entre −1 et +1.



Prises telles quelles, les valeurs de ANG et SIN(ANG) sont imperceptibles sur l'écran dès que l'unité de mesure est le pixel. Il convient donc de leur affecter un coefficient multiplicateur, une échelle, qui permettra un affichage plus ou moins long, plus ou moins large.
Voici un programme qui n'utilise que l'instruction graphique et affiche deux arches de sinusoïdes dans des dimensions compatibles avec l'écran haute résolution défini par le téléviseur. Comme ça ne coûte qu'une ligne, nous y avons ajouté une courbe cosinusoïdale.

```
10 CLS
20 PI=3.14 : EX=15 : EY=50
30 FOR ANG=0 TO 4*PI STEP 1/EX
40 PSET(ANG*EX,SIN(ANG)*EY+EY
50 PSET(ANG*EX,COS(ANG)*EY+EY
60 NEXT ANG
```



Opérations logiques

Le MO6 est un appareil d'une logique implacable, inutile de le rappeler ici. La logique est une branche des mathématiques dont les ingrédients élémentaires sont le vrai et le faux. La cuisine logique consiste à opérer sur ces deux notions d'une façon régulière et rigoureuse. Le mathématicien Boole l'avait déjà compris au siècle dernier en remplaçant vrai et faux par des nombres (0 et 1) à partir desquels il construisit une algèbre dite "de Boole", c'est-à-dire des règles de calcul. Les informaticiens se sont naturellement précipités sur l'algèbre de Boole car, ça tombait bien, 0 et 1 sont pour eux aussi la matière première. Il n'est pas question ici de refaire toute l'histoire mais de mettre le doigt sur quelques aspects importants de la logique des micro-ordinateurs grâce au langage BASIC.

L'opération AND

AND signifie ET. Il doit être compris classiquement. Si A et B sont deux propositions, la proposition "A ET B" est vraie lorsque A et B sont conjointement vérifiées. On construit ainsi une table de vérité de l'opération AND.

AND	F	V
F	F	F
V	F	V

F AND F = F
F AND V = F
V AND F = F
V AND V = V

Module 35 Les opérations logiques (suite)

Votre MO6 représente Vrai par --1 et Faux par 0.
On peut généraliser l'opération logique AND à deux nombres quelconques.

Une opération AND entre deux nombres quelconques. Cette généralisation se fait d'une façon très simple. On peut d'ailleurs l'essayer en mode direct.

```
PRINT -1 AND 0
0
OK
PRINT -1 AND -1
-1
OK
PRINT 53 AND 56
48
OK
```

Les deux premiers calculs confirment les règles opératoires de AND. Par contre, le troisième est proprement stupéfiant! Comment la machine s'y prend-elle pour affirmer que 53 AND 56 = 48?
Pour percer cet épais mystère, il suffit de jeter un coup d'œil sur la représentation binaire des nombres 53 et 56. On s'aperçoit alors que l'opération binaire AND a été faite sur chacun des chiffres des deux nombres (voir annexe 3).

53 →	1	1	0	1	0	1	
56 →	1	1	1	0	0	0	AND
	1	1	0	0	0	0	← 48

Autres opérations logiques

Il existe d'autres opérations logiques construites sur d'autres règles mais qui obéissent toutes aux mêmes lois de généralisation pour les nombres autres que 0 et +1.

A	B	A AND B	A OR B	A IMP B	A XOR B	A EQV B
0	0	0	0	-1	0	1
0	-1	0	-1	-1	1	0
-1	0	0	-1	0	-1	0
-1	-1	-1	-1	1	0	-1

AND : et
OR : ou
IMP : implique
XOR : ou exclusif
EQV: équivalent à

Tests logiques

Les opérations précédentes peuvent être utilisées dans des tests et simplifier la programmation.

Expressions

En BASIC, une expression est une suite de caractères où peuvent intervenir des constantes, des noms de variables et des signes d'opérations. L'interpréteur évalue ces expressions et l'ordre PRINT affiche le résultat de cette évaluation, nous y sommes bien habitués maintenant.

Cette notion d'expression peut être étendue à des écritures du type A = B ou A>B. Dans un test utilisant une telle expression, l'interpréteur procède à une évaluation numérique qui permettra de décider si le test est vérifié ou non. Un nombre est associé à vrai, un autre à faux.

VRAI	FAUX
-1	0

```
A=5 : B=7
PRINT A=B
0
OK
PRINT A<B
-1
OK
PRINT A>=B
0
OK
```

L'évaluation numérique des expressions utilisant une relation de comparaison est faite en secret par l'interpréteur BASIC. Il est inutile que vous en connaissiez tous les arcanes mais vous pouvez imaginer facilement qu'elle se fait en comparant les contenus des variables, bit par bit.

A partir de là, l'utilisation des opérations logiques (AND, OR, NOT) devient une simple affaire de calcul entre des 0 et des −1.

```
A=5 : B=6 : C=7
PRINT A<B AND B<C
-1
OK
PRINT A<B AND B>C
0
OK
PRINT A<B OR B>C
-1
OK
PRINT A>B OR B>C
0
OK
```

Tests logiques

L'opérateur pourra donc utiliser les opérations AND, OR, NOT, etc. pour tester plusieurs conditions derrière un seul IF. A priori, il n'y a aucune restriction sur la longueur des expressions et le nombre d'opérations sinon celle de la limite du nombre de caractères dans une ligne d'instructions.

Affectation logique

Si vous avez rédigé consciencieusement une série de petits programmes, vous avez sans doute remarqué que l'interpréteur BASIC est intraitable en ce qui concerne la syntaxe. La moindre ponctuation absente ou mal placée, la moindre règle non respectée provoquent un message d'erreur et l'interruption du programme.
En ce qui concerne l'affectation (=), vous n'avez pas le droit de faire n'importe quoi. Une instruction comme A=3 est correcte mais 3=A ne l'est pas. Que pensez-vous de A = B = C?
Contrairement à ce que vous pourriez croire un peu vite, cette instruction est tout à fait correcte. Elle a même un sens très précis et, à l'occasion, peut servir.
Le signe = après A est un signe d'affectation. L'expression à droite de ce signe doit être évaluée et sa valeur rangée dans la variable A. Cette expression est B = C, sa valeur est 0 ou −1 suivant que B est ou non égal à C. La valeur de A donnera après cette instruction une indication sur l'égalité de B et C.

```
B=5 : C=6
A=B=C : PRINT A
0
OK
B=10 : C=10
A=B=C : PRINT A
-1
OK
```

Module 36 Taille mémoire

Avant de poursuivre, abordons ici un problème qui hante les nuits des programmeurs avertis: celui de la mémoire libre.
La MEV a pour chaque modèle de micro-ordinateur une capacité propre mesurée en kilo-octets. Chaque instruction d'un programme, chaque variable occupe un peu de mémoire. Lorsque le programme grossit, la mémoire libre diminue. A un certain moment, il n'en reste plus, c'est l'impasse, le programmeur est désespéré.

La commande PRINT FRE(0) indique à tout moment la capacité de mémoire vive disponible. (FRE est mis pour FREE, libre en anglais.) Dans l'exemple suivant, nous testons le nombre d'octets disponibles avant l'écriture d'un programme sans commentaires, puis après l'addition de commentaires.
Allumez l'ordinateur.

```
PRINT FRE(0)
110120
OK
20 A=19 : B=38
30 C=A+B
40 PRINT C
RUN
57
OK
PRINT FRE(0)
110089
OK
10 REM SOMME
35 REM AFFICHAGE SOMME
RUN
57
OK
PRINT FRE(0)
110055
OK
NEW
PRINT FRE(0)
110120
OK
```

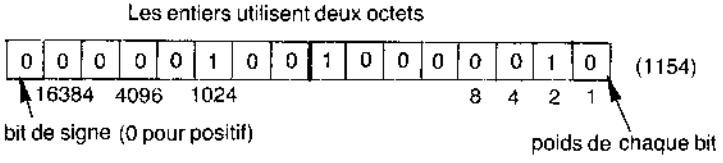
Conclusions:
1. Au départ, 110120 octets libres.
2. Après déroulement du programme sans commentaires, 110089 octets libres, soit 31 octets pour le programme.
3. Après déroulement du programme commenté, 110055 octets libres, soit 34 octets pour les commentaires.
4. Après NEW, on récupère les 110120 octets du départ.

La conclusion est claire: les commentaires prennent de la place... On s'en serait douté!

Un peu, beaucoup, pas du tout de REM?
Nous nous garderons de tout dogmatisme en la matière. A tel point que nous nous contenterons de cet unique commentaire sans prendre de très gros risques: mieux vaut un REM court et bien placé que beaucoup de REM inutilisés, verbeux et mal placés.
Toute intervention, toute commande, "tout" prend de la place mémoire...

Mémoires occupées par les variables

Pour stocker une variable en mémoire, BASIC utilise un certain nombre d'octets pour : le nom de la variable, son type, son adresse et sa valeur.



Quinze bits permettent de coder un nombre entre 0 et 32767 (c'est-à-dire $2^{15}-1$). Le seizième bit code le signe.
Le nombre codé ci-dessus en exemple est le nombre 1154 (1024+128+2); son bit de signe (positif) vaut 0.
Si le bit de signe vaut 1, le nombre représenté est négatif; sa valeur absolue est obtenue en changeant tous les bits (0 en 1 et 1 en 0) et en ajoutant 1 au résultat. (Cette opération s'appelle le complément à 2.)
Voici par exemple la représentation de −2:

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0

Les réels utilisent quatre octets. Un réel se présente extérieurement sous la forme:

0.123456 E 30
mantisse exposant

La mantisse est stockée à l'aide de trois octets où un bit est utilisé pour le signe. On obtient ainsi les sept chiffres décimaux significatifs en simple précision.
L'exposant occupe un octet, dont un bit pour le signe, et permet de coder jusqu'à 127. La limite de représentation d'un entier positif est donc de 2^{127}, soit environ 10^{38}.

Les chaînes de caractères utilisent un octet par caractère stocké plus trois octets pour la gestion de la chaîne.

Pour les tableaux, chaque élément occupe la place prévue par son type; il faut ajouter quelques octets pour les caractéristiques du tableau.

Mémoire d'écran

La mémoire d'écran est une "mémoire vive" car son contenu peut être modifié à tout instant, soit par l'utilisateur, soit par le micro-ordinateur lui-même. L'écran est composé de 64 000 points ou pixels (200×320). La mémoire d'écran réserve un emplacement mémoire pour chaque pixel de l'écran de telle façon qu'on peut imaginer la mémoire d'écran comme une copie conforme de l'écran où les pixels allumés sont remplacés par des 1 et les pixels éteints par des 0.
Si chacun des 64 000 points pouvait être colorié de l'une des seize couleurs, il faudrait utiliser 32 K-octets pour mémoriser tous ces renseignements. Pour économiser la place mémoire, on a ramené ce stockage sur 16 K-octets. La contrepartie, c'est qu'il n'est pas possible de traiter chaque point séparément des autres: la conséquence est "l'effet bavure" que nous avons déjà souligné lors d'exemples graphiques.
L'écran est donc stocké sous forme de segments de huit points.

Pour chaque segment:
— Un octet indique l'appartenance de chacun des points à la catégorie "écriture" ou à la catégorie "fond".

1	0	1	1	0	0	1	0

point d'écriture (1) point de fond (0)

— Un octet, divisé en 2×4, précise la couleur du fond et la couleur de l'écriture.

0	0	1	1	1	0	1	1

couleur d'écriture (ici 3 écrit en binaire)
couleur du fond (ici 1+2+8=11 en binaire)

Sur 4 bits, on peut repérer 2^4, c'est-à-dire 16 couleurs.

Module 37 Descente ASCII

Dans le module précédent, on rappelait qu'un caractère codé dans la mémoire de l'ordinateur occupe un octet. Un octet, c'est huit bits, soit la possibilité de coder $2^8 = 256$ caractères différents. En fait, sur les huit bits offerts, seuls sept servent effectivement au codage. Cela donne $2^7 = 128$ caractères codés par l'ASCII dont pour l'instant nous ne connaissons que les lettres (voir module 10):

A est codé par 65 en décimal, soit 01000001 sur un octet

Z est codé par 90 en décimal, soit 01011010 sur un octet

Examinons de plus près le code ASCII grâce à CHR$.
Cette descente aux enfers du codage informatique est guidée et nous vous proposons de l'effectuer en deux temps.
— Premier temps: ne pas descendre en dessous du code 32.

```
FOR I=32 TO 127 : PRINT CHR$(I) : NEXT I
```

et vous voyez défiler tous les caractères affichables disponibles sur votre MO6.

— Deuxième temps: si vous descendez entre 0 et 31... gare aux surprises et au "plantage" de la machine.
En effet, les codes inférieurs à 32 sont réservés à l'usage de la machine. On y trouve les codes du retour chariot, des touches de déplacement du curseur, du clignotement, de la sonnette, etc. Ces caractères ne sont pas affichables à l'écran et risquent d'y provoquer une jolie pagaille.
Dans le module 43, nous vous montrerons une application de l'utilisation du code ASCII pour gérer le déplacement du curseur.

PEEK et POKE

Tout le travail du programmeur ou de l'utilisateur d'un micro-ordinateur consiste à lire et modifier le contenu de certaines cases mémoire, à ranger et à reprendre.

Ranger

Une instruction très simple en BASIC permet de ranger dans une case mémoire, un nombre entier compris entre 0 et 255. Deux renseignements sont indispensables pour réaliser cette manœuvre: quoi ranger et où le ranger, autrement dit quel octet et à quelle adresse. L'ordre de rangement est POKE suivi dans l'ordre, de l'adresse et du nombre à ranger.

POKE l'adresse,le nombre à ranger

```
POKE 17400,178
OK
```

Cet OK vous assure que la manœuvre de rangement a bien été exécutée mais puisque aucun ordre d'affichage n'est donné, rien ne s'affiche: tout s'est passé dans les entrailles de la machine.

Reprendre

Nous imaginons que vous brûlez de vérifier que votre ordre a bien été exécuté. Vous le pouvez grâce à l'ordre PEEK qui va "piquer" dans la case mémoire de votre choix.
Pour obtenir l'affichage de la case mémoire d'adresse 17400, il faut ordonner d'afficher (PRINT) le contenu (PEEK) de l'adresse 17400.

```
PRINT PEEK(17400)
178
OK
```

PEEK lit sans le modifier le contenu de la case mémoire. Seul POKE peut réaliser des modifications.

Peeker et poker partout

Soient deux expériences qui vous permettront de saisir une différence essentielle entre mémoire morte (MEM) et mémoire vive (MEV).

Première expérience

```
PRINT PEEK(41000)
76
OK
POKE 1000,42
OK
PRINT PEEK(1000)
76
OK
```

Résumons:
Dans la case mémoire d'adresse 41000, j'ai trouvé le nombre 76. J'ai essayé de le remplacer par 42 grâce à un POKE. Rien à faire, c'est toujours 76 qui s'y trouve. L'ordre POKE a été ignoré. Il faut en conclure que la case mémoire d'adresse 1000 appartient à la mémoire morte qui reste inviolable: son contenu ne peut pas être modifié par l'utilisateur.

Deuxième expérience

```
PRINT PEEK(17400)
0
OK
POKE 17400.42
OK
PRINT PEEK(17400)
42
OK
```

17400 est sans conteste une adresse de mémoire vive. Au départ, toutes les cases mémoire de la MEV, exceptée la mémoire d'écran, contiennent 0. C'est l'utilisateur qui les remplira progressivement.

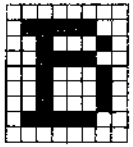
Toute la programmation pourrait être faite avec des PEEK et des POKE... ce serait extrêmement fastidieux. Heureusement, le BASIC offre des commodités! Il est néanmoins important que vous ne perdiez pas de vue qu'au fond, il s'agira toujours de ranger et de reprendre des nombres dans des cases mémoire. C'est ce que la machine fera quoi qu'il arrive. Les facilités offertes par un langage comme le BASIC consistent à laisser la machine choisir elle-même les cases mémoire de rangement.

Avertissement
En faisant des POKE n'importe où dans la mémoire du MO6, vous vous exposez à un double risque : l'ennui, d'abord qui résulte normalement d'activités sans objectif, le risque ensuite de "planter" votre MO6. Pas d'inquiétude cependant, en appuyant sur RESET vous remettez tout en ordre. Quoiqu'il en soit, les auteurs de ce manuel ainsi que les constructeurs de la machine déclinent toute responsabilité...

Module 38 Affichage d'un caractère

Lorsque l'utilisateur appuie sur une touche du clavier, il a toutes les raisons d'espérer l'affichage sur l'écran du signe inscrit sur la touche qu'il enfonce. Il est intéressant de se demander ce qui se passe réellement entre l'appui de la touche et l'affichage sur l'écran. On peut se persuader facilement qu'entre ces deux étapes, rien de ce qui peut ressembler au caractère affiché n'a circulé dans les fils du micro-ordinateur! Ici encore, c'est le codage binaire qui impose sa loi. Prenons l'exemple de la lettre "B", comme BASIC. Elle peut être affichée de différentes façons sur l'écran mais le principe est toujours le même: éclairement de points bien choisis sur une grille.

affichage de B
sur une grille 8×8



On peut observer que le cadre extérieur de la grille n'est jamais rempli. Cette précaution vise à permettre un affichage agréable des caractères les uns à côté des autres et les uns au-dessus des autres: deux lignes ou deux colonnes de pixels au moins séparent toujours deux caractères voisins. Pour bien comprendre les principes qui gouvernent la transmission de l'information entre le clavier et l'écran, il est tout à fait naturel de remplacer dans la grille d'affichage du caractère B les points allumés par des 1 et les points éteints par des 0.

traduction binaire
de l'affichage d'un B
sur une grille 8×8

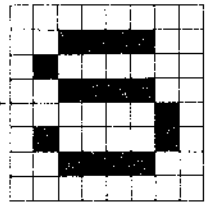
0	0	0	0	0	0	0	0
0	1	1	1	1	1	0	0
0	0	1	0	0	0	1	0
0	0	1	1	1	1	0	0
0	0	1	0	0	0	1	0
0	0	1	0	0	0	1	0
0	1	1	1	1	1	0	0
0	0	0	0	0	0	0	0

Chaque ligne de cette grille peut être considérée comme un octet et par conséquent, la grille tout entière est parfaitement codée par un ensemble de huit octets.
Pour le caractère B, il s'agit, dans l'ordre des octets :00000000, 01111100, 00100010, 00111100, 00100010, 00100010, 0111100, 00000000.
Pour faciliter la compréhension, on peut donner la traduction décimale de chacun de ces octets, si bien que l'affichage d'un caractère se code par un ensemble de huit nombres compris entre 0 et 255. Il s'agit d'un bon codage, c'est-à-dire que l'affichage du caractère sur la grille détermine de façon unique l'ensemble des huit nombres mais inversement, et c'est plus intéressant, la connaissance des huit nombres permet de retrouver exactement la grille d'affichage.

Exemple:

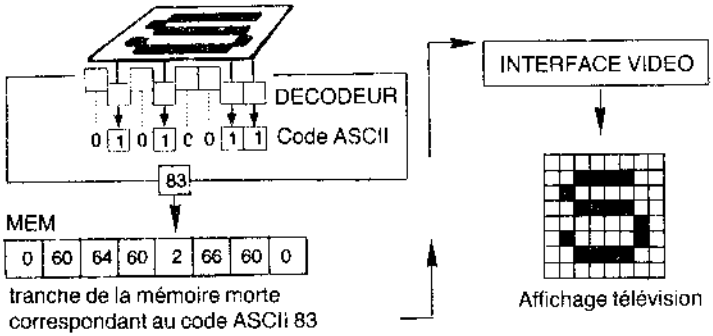
Quel est le caractère codé par les nombres: 0, 60, 64, 60, 2, 66, 60, 0?
Chacun de ces nombres a une traduction binaire unique qui permet de retrouver le caractère.

0 →	0	0	0	0	0	0	0	0
60 →	0	0	1	1	1	1	0	0
64 →	0	1	0	0	0	0	0	0
60 →	0	0	1	1	1	1	0	0
2 →	0	0	0	0	0	0	1	0
66 →	0	1	0	0	0	0	1	0
60 →	0	0	1	1	1	1	0	0
0 →	0	0	0	0	0	0	0	0



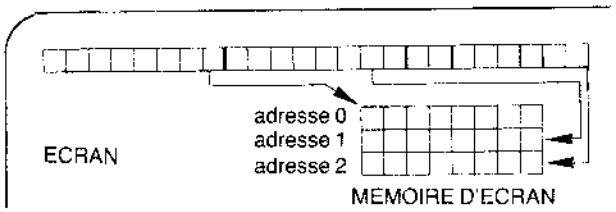
Pour permettre à l'ordinateur d'afficher correctement tous les caractères du clavier, il suffit de bien ranger dans la mémoire morte, et par groupe de huit, les octets correspondant à chaque caractère. Pour retrouver la tranche de MEM concernée par la touche appuyée, il existe un décodeur ASCII des touches de clavier. Le schéma suivant résume l'ensemble de la manœuvre. Nous avons imaginé un

décodeur mécanique pour simplifier la situation et parce que cela nous amusait, mais le lecteur doit savoir qu'en réalité, il s'agit d'un montage électronique. Le code ASCII du caractère S est 83.

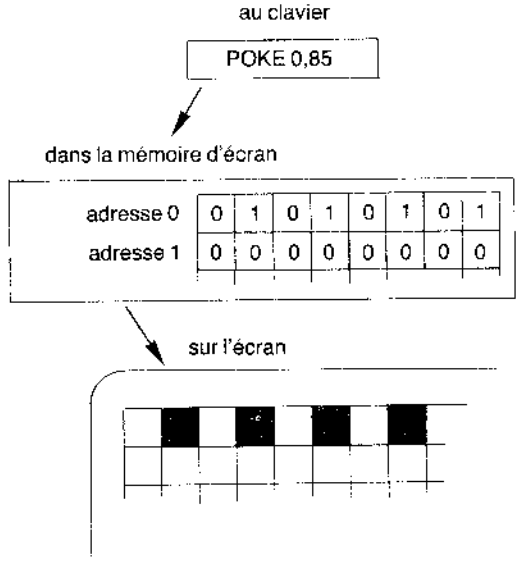


Agir sur la mémoire écran

La mémoire écran est une partie de la mémoire vive. Rien n'empêche donc l'utilisateur de modifier le contenu de certaines cases mémoire pour voir le résultat à l'écran. Il suffit de connaître l'adresse. Pour le MO6, la mémoire d'écran est tout en bas de la MEV: la première adresse est 0, la dernière est 7999.
Le principe d'organisation de cette mémoire est très simple. Le premier octet correspond aux huit premiers points horizontaux en haut à gauche, le deuxième (adresse 1) aux huit points suivants, etc. Quand on arrive au bout d'une ligne, on passe tout naturellement au début de la suivante.



Le principe de correspondance entre les bits de la mémoire et les pixels de l'écran est des plus simples : 0 pour un pixel éteint, 1 pour un pixel allumé.
Choisissons de placer le nombre 85 dans la case mémoire d'adresse 0. Dans le système binaire, 85 s'écrit 01010101.



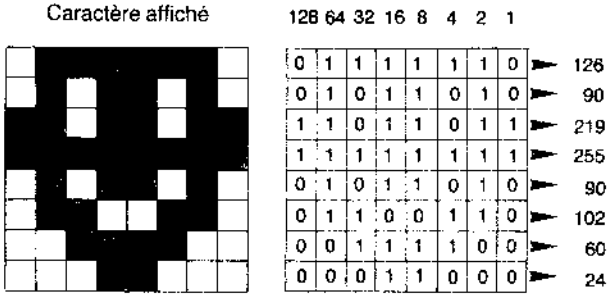
Pour connaître la place d'une case mémoire d'adresse X sur l'écran, il suffit de taper l'instruction:

```
POKE X,255
```

et d'observer à quel endroit précis de l'écran la petite barrette de huit pixels apparaît. Un excellent jeu consisterait à essayer de le prévoir. Pour effacer les points allumés, il suffit de faire un POKE dans la case mémoire concernée en y plaçant le nombre 0.

Module 39 Définir ses propres caractères

Les caractères courants (ceux du code ASCII) sont stockés dans la mémoire morte et sont intouchables.
De son côté, l'utilisateur dispose d'une partie de la mémoire vive pour y créer ses propres caractères. Une instruction (CLEAR) lui permettra de réserver un certain nombre de cases mémoire pour créer ces caractères ; une autre (DEFGR$) ira les ranger dans la partie de mémoire vive réservée.
On utilise toujours une grille 8×8 pour dessiner un caractère ainsi que la traduction par des 1 des parties allumées.
Choisissons une face joviale pour nous donner du courage.



La mémorisation de ce caractère spécial s'effectue en trois temps.

CLEAR,,1
Cette instruction annonce à la machine que nous allons lui proposer un nouveau caractère (CLEAR,,5 pour 5 caractères, etc.).

DEFGR$(0)=126,90,219,255,90,102,60,24
Voici donc l'instruction qui définit numériquement notre face joviale.

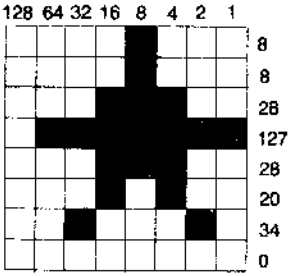
— DEF pour définir,
— GR pour graphique,
— $ pour caractère,
— 0 est le numéro de notre face joviale.

Qu'on définisse un ou plusieurs caractères, la numérotation doit obligatoirement commencer par 0 et se poursuivre dans l'ordre des entiers naturels jusqu'à 127 au maximum.
Vous reconnaîtrez facilement, après le signe = , les 8 nombres décimaux constituant le codage numérique de notre face joviale.

PRINT GR$(0)
Cette instruction provoquera l'affichage du caractère GR$ (0). Il est possible, comme pour tout autre caractère, de fixer sa position par LOCATE, sa taille par ATTRB, sa couleur par COLOR. On peut changer son nom : A$ = GR$ (0) et c'est A$ qui contient la face joviale. B$ = GR$ (0) + GR$ (0) et B$ contient une double face joviale.

L'art et la bannière

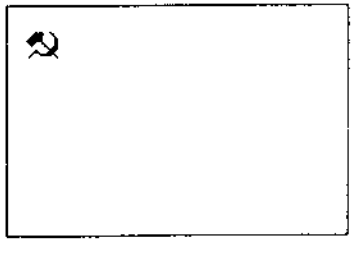
Nous ne ferons pas l'injure au lecteur de lui proposer un programme dessinant le drapeau français. Non pas que nous reniions notre douce patrie mais trois BOXF bleu, blanc, rouge ne méritent pas qu'on s'y attarde. Notre cœur de programmeur a succombé en revanche devant les bannières des deux super-grands.
Voici un programme qui dessine le drapeau américain et utilise un caractère défini par l'utilisateur, l'étoile.



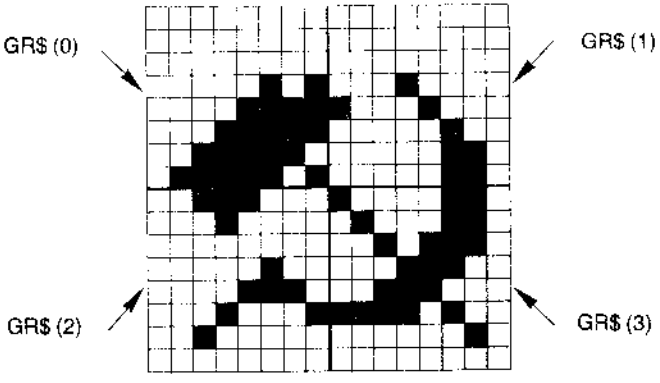

```
10 ' DRAPEAU AMERICAIN
20 SCREEN 7,4,0 : CLS : LOCATE 0,0,0
```

```
30 CLEAR, , 1
40 DEFGR$(0)=8,8,28,127,28,20,34,0
50 FOR X=0 TO 10
60 FOR Y=0 TO 8
70 IF (X+Y)MOD 2=0 THEN LOCATE X,Y : PRINT GR$(0)
80 NEXT Y
90 NEXT X
100 X=90
110 FOR Y=0 TO 132 STEP 11
120 IF Y>72 THEN X=0
130 C=1+6*(Y MOD 2)
140 BOXF(X,Y)-(220,Y+10),C
150 NEXT Y
160 LOCATE 0,20 : END
```

Commentaires :
50-90. Deux boucles imbriquées pour imprimer les 50 étoiles.
Le test sur la valeur de (X+Y)MOD 2 permet de reconnaître les cases où ces étoiles doivent être présentes (quinconce).
130. Formule (un peu compliquée !) pour déterminer la couleur 1 ou 7.



Il nous faut maintenant tenter de définir le caractère "faucille et marteau".
Pour obtenir une représentation suffisamment réaliste, nous allons utiliser quatre caractères.




```
10 ' DRAPEAU SOVIETIQUE
20 SCREEN 3,0,0 : CLS
30 CLEAR, , 4
40 DEFGR$(0)=0,0,0,5,15,31,63,127
50 DEFGR$(1)=0,0,0,16,136,4,6,6
60 DEFGR$(2)=56,16,0,4,14,17,32,0
70 DEFGR$(3)=134,70,46,28,56,244,2,0
80 FM$=GR$(0)+GR$(1)+CHR$(10)+CHR$(8)+CHR$(8)+GR$(2)+GR$(3)
90 BOXF(0,0)-(150,100),-2
100 LOCATE 1,1,0 : COLOR 3,1 : PRINT FM$
110 LOCATE 0,20 : END
```

Commentaires : Pour imprimer le motif en une seule instruction, on a recours à un stratagème : on utilise les codes du déplacement du curseur :
— PRINT CHR$(10) provoque un déplacement vers le bas d'une ligne.
— PRINT CHR$(8) provoque un déplacement vers la gauche d'une colonne.
(Voir aussi le module 43)

Module 40 Contrôle des entrées

Si le programme que vous envisagez d'écrire est interactif, vous utiliserez soit l'instruction INPUT, soit INKEY$.
Si vous choisissez INPUT, on peut conclure deux choses :
— Il ne s'agit pas d'un jeu rapide.
— Vous faites totalement confiance à l'utilisateur.
Avec INPUT, vous ne pourrez contrôler la suite des touches frappées qu'après l'appui de la touche ENTRÉE. L'utilisateur prendra le temps qu'il voudra, mais plus grave, il tapera tous les caractères qu'il voudra et autant qu'il le voudra.
Tout porte donc à s'intéresser plutôt à INKEY$ qui ne présente pas les inconvénients de INPUT que nous venons de souligner. En revanche, deux inconvénients apparaissent lors d'une prise sous INKEY$, l'un mineur, l'autre majeur. L'inconvénient mineur concerne l'absence d'affichage. On y remédie facilement si besoin est en faisant suivre les tests d'un ordre PRINT.
En revanche, avec INKEY$, on perd l'avantage de l'éditeur présent sous INPUT. Avec cette dernière instruction, l'utilisateur a en général la possibilité de ramener le curseur en arrière pour corriger certains caractères avant de valider.
Nous proposons ci-dessous une procédure qui enregistre une série de chiffres, en contrôlant à chaque étape qu'il s'agit bien de chiffres. L'utilisateur aura la possibilité d'aller corriger un chiffre déjà inscrit en ramenant le curseur en arrière, ce qui aura pour effet d'effacer les chiffres sur lesquels il passe. Le nombre maximum de chiffres peut être fixé à l'avance.

```
1 '
2 ' EDITEUR INKEY$
3 '
10 LMAX=10
20 PRINT "Donnez un nombre entier de moins de" ; LMAX ; "chiffres :" ;
30 A$="" : L=0 : GOSUB 1000
999 END
1000 '
1001 ' Saisie clavier
1002 '
1010 R$=INKEY$
1020 IF R$="" THEN 1010
1030 C=ASC(R$)
1040 IF C=13 THEN 1999
1050 IF C=8 AND L>0 THEN A$=LEFT$(A$,L-1) : PRINT
CHR$(8)+" "+CHR$(8) ; : L=L-1 : GOTO 1010
1060 IF R$<"0" OR R$>"9" THEN 1010
1070 IF L=LMAX THEN 1010
1080 PRINT R$ ; : A$=A$+R$ : L=L+1
1090 GOTO 1010
1999 RETURN
```

Variables

A$ est la chaîne qui recevra le nombre. L est sa longueur.
LMAX est la longueur maximale de cette chaîne.
R$ est la touche tapée éventuellement au clavier.
C est le code ASCII de R$.

Commentaires :
1020 Si aucune touche n'est appuyée, il faut attendre...
1040 Si la touche appuyée est ENTREE, on sort du sous-programme.
1050 Si la touche appuyée est le retour de curseur, et si A$ n'est pas vide, alors on enlève le dernier caractère dans A$ (correction) puis on recule le curseur, on imprime un blanc et on recule le curseur.
1060 Si la touche appuyée n'est pas un chiffre, on retourne à l'enregistrement.
1070 Si A$ est plein, on retourne à l'enregistrement.
1080 Enfin ! On imprime le chiffre choisi, on l'ajoute à A$ et on retourne à l'enregistrement.

Pianoter au clavier ONKEY=...GOTO... ou ONKEY=...GOSUB...

Le BASIC offre la possibilité à l'utilisateur de définir certaines actions à partir du clavier. C'est-à-dire que l'appui d'une touche provoquera automatiquement le déroulement d'une procédure prévue à cet effet. Le programme doit bien évidemment être prévenu de telles interventions : c'est le rôle des instructions ONKEY=...GOTO... et ONKEY=...GOSUB...
Construisons donc un clavier musical. Choisissons par exemple de faire correspondre les chiffres de 1 à 0 aux sept notes de l'octave 5 et aux trois dernières notes de l'octave 4 (sur le clavier, 0 est la touche située à droite de 9).

1 correspond à O4 SO, ..., 9 correspond à O5 LA, 0 correspond à O5 SI

En tête de programme, on déduit les touches dont les actions sont attendues. Elles provoquent un branchement par GOTO.

Programme

```
5 ' PIANO CLAVIER
10 ONKEY="1" GOTO 210
20 ONKEY="2" GOTO 220
30 ONKEY="3" GOTO 230
40 ONKEY="4" GOTO 240
50 ONKEY="5" GOTO 250
60 ONKEY="6" GOTO 260
70 ONKEY="7" GOTO 270
80 ONKEY="8" GOTO 280
90 ONKEY="9" GOTO 290
100 ONKEY="0" GOTO 300
200 GOTO 200
210 PLAY "O4 SO" : GOTO 200
220 PLAY "O4 LA" : GOTO 200
230 PLAY "O4 SI" : GOTO 200
240 PLAY "O5 DO" : GOTO 200
250 PLAY "O5 RE" : GOTO 200
260 PLAY "O5 MI" : GOTO 200
270 PLAY "O5 FA" : GOTO 200
280 PLAY "O5 SO " : GOTO 200
290 PLAY "O5 LA" : GOTO 200
300 PLAY "O5 LA" : GOTO 200
```

Commentaire : l'instruction d'attente est la boucle sans fin
200 GOTO 200

Remarque : ce genre de programme fait apprécier la possibilité de dupliquer les lignes en changeant leur numéro.

Le branchement sur un sous-programme peut se faire aussi au moyen des touches du clavier avec ONKEY...GOSUB.

Module 41 Menu

Les meilleurs restaurants ne sont pas ceux qui présentent le plus beau menu : ce qui compte, c'est ce qu'il y a dans l'assiette.
Il en est de même avec les programmes : les bons menus ne font pas la bonne cuisine mais un bon programme ne saurait se passer d'un menu bien présenté.
La plupart des programmes étoffés (jeux ou pas) commencent et finissent par un menu. Le menu d'entrée propose des options (niveaux, choix dans un catalogue) et le menu de sortie est une alternative entre la poursuite ou la fin de l'activité.
En général, ces deux menus sont placés dans des sous-programmes en fin de listing afin de bien les dégager du programme principal et de favoriser la lisibilité. L'une des premières instructions du programme principal envoie au menu initial et l'une des dernières au menu final.
Il est tout à fait légitime de commencer par construire ces parties du programme, de façon à pouvoir les essayer, et de construire l'ensemble autour de blocs bien numérotés.
Voici, pour l'exemple, un programme fort ambitieux puisqu'il propose au menu quatre jeux de cartes : bataille, belote, poker et bridge. Seuls les menus de début et de fin sont écrits entièrement. Les sous-programmes de jeu sont fixés de telle façon que le programme peut être testé (il tourne) : il ne reste qu'à faire la cuisine, les plats sont propres.

```
1 '
2 ' JEUX DE CARTES
3 '
20 GOSUB 10000
30 ON CHOIX GOSUB 1000,2000,3000,4000
40 GOSUB 15000
50 ON ZZ GOTO 20,999
999 END
1000 '
1001 ' BATAILLE
1002 '
1010 CLS : PRINT "BATAILLE"
1020 GOSUB 20000
1999 RETURN
2000 '
2001 ' BELOTE
2002 '
2010 CLS : PRINT "BELOTE"
2020 GOSUB 20000
2999 RETURN
3000 '
3001 ' BRIDGE
3002 '
3010 CLS : PRINT "BRIDGE"
3020 GOSUB 20000
3999 RETURN
4000 '
4001 ' POKER
4002 '
4010 CLS : PRINT "POKER"
4020 GOSUB 20000
4999 RETURN
10000 '
10001 ' MENU
10002 '
10010 CLS : PRINT "JEU DE CARTES":PRINT
10020 PRINT "1 : BATAILLE"
10030 PRINT "2 : BELOTE"
10040 PRINT "3 : BRIDGE"
10050 PRINT "4 : POKER"
10060 GOSUB 20000
10070 IF ZR<49 OR ZR>52 THEN 10060
10080 CHOIX=ZR-48
10999 RETURN
15000 '
15001 ' FIN
15002 '
15010 CLS : PRINT "VOULEZ-VOUS CONTINUER?"
15020 GOSUB 20000
```

```
15030 IF ZR=79 THEN ZZ=1 : GOTO 15999
15040 IF ZR=78 THEN ZZ=2 : GOTO 15999
15050 GOTO 15020
15999 RETURN
20000 '
20001 ' SAISIE CLAVIER
20002 '
20010 ZR$=INKEY$
20020 IF ZR$="" THEN 20010
20030 ZR=ASC(ZR$)
20999 RETURN
```

Entrée par crayon optique

Un menu interactif peut proposer des réponses à l'écran sous forme de zones à pointer à l'aide du crayon optique.
Le couple d'instructions PEN et ON PEN... permet :
— de définir jusqu'à huit cases différentes à l'écran,
— de demander l'exécution d'une ligne différente pour chaque case lorsqu'une case est désignée au crayon optique.
Après ON PEN, on peut trouver un branchement à une ligne d'instruction par GOTO ou à un sous-programme par GOSUB. Le principe de fonctionnement est le même que pour ON...GOTO et ON...GOSUB.

PEN

Cette instruction permet de mettre en relation un numéro compris entre 0 et 7 avec une zone définie à l'écran.

Exemple de programme :
— Un premier sous-programme affiche les textes des choix proposés et les cases destinées à recevoir les réponses du crayon optique.
— Chaque réponse est traitée dans un sous-programme spécialisé.

Attention : si le crayon optique vise une zone extérieure aux boîtes définies par le premier sous-programme, l'instruction ON PEN aura pour effet de passer à l'instruction suivante.

Programme

```
10 '
20 CLS : SCREEN 0,6,6
30 LOCATE 3,11,0
40 GOSUB 1000
50 ON PEN GOSUB 2000,3000,4000,5000,6000,7000,8000,9000
60 GOTO 50
999 END
1000 REM MENU PRINCIPAL
1010 ' Texte des choix proposés
1100 FOR N=0 TO 7
1110 COL=20+N*24
1120 BOX(COL,80)-(COL+24,95)
1130 PEN N : (COL,80)-(COL+23,95)
1140 NEXT N
1999 RETURN
2000 REM
2999 RETURN
3000 REM
3999 RETURN
4000 REM
4999 RETURN
5000 REM
5999 RETURN
6000 REM
6999 RETURN
7000 REM
7999 RETURN
8000 REM
8999 RETURN
9000 REM
9999 RETURN
```

Il ne vous reste plus qu'à remplir les sous-programmes.

Module 42 Tableau soigné

Après les entrées aux petits oignons, voici les orties.
La présentation de tableaux numériques dont les éléments sont des résultats de calcul pose des problèmes de présentation et plus précisément, d'alignement si l'on veut que les nombres rentrent dans les cases d'une grille fixe et que les virgules (points décimaux) soient alignées. Une instruction BASIC classique règle ce genre de difficultés, c'est PRINTUSING.
L'instruction PRINTUSING doit être suivie d'une chaîne de caractères définissant le format d'impression, puis du nombre à imprimer.

Format

Le format "# # # . # #" permet d'afficher tous les nombres avec trois chiffres avant le point décimal et deux chiffres après.

```
A$="###.##"
PRINT USING A$;453.2
453.20
PRINT USING A$;27.438
 27.44
PRINT USING A$;5845.3
%5845.30
OK
```

Commentaires :
— Eventuellement, la partie décimale est complétée avec des 0.
— Si tous les chiffres de la partie décimale ne sont pas affichés, il y a arrondi.
— Si la partie entière est trop grande, le nombre est affiché précédé du signe % qui indique une incohérence.

Exemple : Ce programme construit un tableau dans lequel figurent un prix hors taxe, le montant de la taxe (TVA) et le prix toutes taxes comprises (TTC).
La saisie des données est faite dans un sous-programme simple (INPUT) qu'il est possible de durcir.

```
1 '
2 ' TVA
3 '
20 A$="###.##"
30 GOSUB 1000 : CLS
40 PRINT TAB(2) ; "PRIX HT" ; TAB(12) : "TVA" ; TAB(22) ; "TTC" . PRINT
50 FOR K=1 TO NA
60 PRINT TAB(2) ; : PRINT USING A$ ; HT(K) ;
70 PRINT TAB(12) ; : PRINT USING A$ ; MT(K) ;
80 PRINT TAB(22) ; : PRINT USING A$;TTC(K)
100 NEXT K
999 END
1000 '
1001 ' SAISIE
1002 '
1010 CLS
1020 INPUT "NOMBRE D'ARTICLES :" ; NA
1030 INPUT "POURCENTAGE TVA :" ; TVA
1040 DIM HT(NA),MT(NA),TTC(NA)
1050 FOR K=1 TO NA
1060 PRINT "ARTICLE :" ; K ;
1070 INPUT "PRIX HORS TAXES :" ; HT(K)
1080 MT(K)=HT(K)*TVA/100
1090 TTC(K)=HT(K)+MT(K)
1100 NEXT K
1999 RETURN
```

Déroulement

PRIX HT	TVA	TTC
78.90	14.60	93.50
175.00	32.38	207.38
567.42	104.97	672.39
89.44	16.55	105.99
715.00	132.28	847.28

Compléments

L'instruction PRINTUSING permet encore bien d'autres fantaisies qu'il serait trop long de détailler ici. Reportez-vous aux Références.

On peut demander dans le format :
— L'affichage d'un signe ou d'un autre caractère au début du nombre.
— L'affichage d'astérisques dans les zones non imprimées.
— L'affichage de virgules pour le découpage des parties entières de plus de trois chiffres (habitude anglo-saxonne).
— L'affichage systématique en notation exponentielle.
PRINTUSING peut également être utilisé pour éditer proprement des chaînes de caractères. Le format "!" n'affiche que le premier caractère.

```
PRINT USING "!";"AL1"
A
OK
```

Le format "@@" (ou "%%") réserve la place pour le nombre d'espaces compris entre les deux @ (ou les deux %) plus deux.

Ecrire partout

Tabulation

L'instruction PRINT TAB (X) permet de décaler l'affichage sur une ligne de X caractères à partir de la première colonne à gauche de l'écran. C'est très utile pour afficher des textes au milieu de l'écran ou aligner plusieurs colonnes de chaînes de caractères.
La première colonne a le numéro 1.
Voici une routine qui affiche des chaînes A$ en les centrant sur l'écran. Le nombre de colonnes NC est ici fixé à 40 en ligne 10.

```
10 NC=40
20 A$="BASIC" : GOSUB 100
30 A$="TABULATION" : GOSUB 100
99 END
100 REM CENTRAGE
110 LONG=LEN(A$)
120 IF LONG>NC THEN X=0 ELSE X=INT((NC-LONG)/2)
130 PRINT TAB(X) ; A$
199 RETURN
RUN
                    BASIC
                TABULATION
```

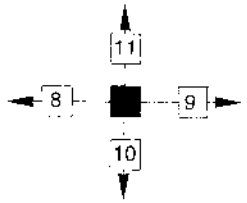
Module 43 Déplacements au clavier

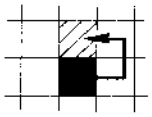
Plutôt que de laisser le programme déplacer tout seul les caractères qu'il veut où il veut, nous proposons un programme interactif qui permettra à l'utilisateur de déplacer un caractère sur l'écran à sa guise.
C'est évidemment une routine indispensable pour les jeux d'action. Avec les quatre flèches du curseur, le joueur déplace son représentant sur l'écran en tentant d'échapper aux horribles dangers qui le menacent.

L'idée de ce programme est la suivante :
1 — Lire la touche appuyée par INKEY$.
2 — Tester si le déplacement demandé ne fait pas sortir de l'écran.
3 — Effacer le caractère puis le réécrire à gauche, à droite, au-dessus ou au-dessous.

Programme

Seules les quatre touches de déplacement du curseur sont autorisées. Les codes ASCII sont 8, 9, 10 et 11.
Les tests 110, 210, 310, 410 sont des tests de bord qui renvoient à l'entrée clavier.
Pour comprendre la signification de D$, il faut regarder dans la ligne d'affichage 500.
CHR$(32) est le blanc qui efface le caractère. D$ permet de situer le curseur à l'endroit qui correspond à la touche frappée. Ainsi, pour la touche vers le haut, D$ fera remonter puis aller un pas à gauche.





```
1 '
2 ' DEPLACEMENT CLAVIER
3 '
20 CLS
30 X$=CHR$(8) : SX=20 : SY=10 : X=0 : Y=0
40 J$=CHR$(127)+X$ : PRINT J$
50 R$=INKEY$
60 IF R$="" THEN GOTO 50
70 A=ASC(R$)
80 IF A<8 OR A>11 THEN GOTO 50
90 ON A-7 GOTO 200,100,400,300
100 '
101 ' DROITE
102 '
110 IF X>SX THEN GOTO 50
120 D$="" : X=X+1 : GOTO 500
200 '
201 ' GAUCHE
202 '
210 IF X<1 THEN GOTO 50
220 D$=X$+X$ : X=X-1 : GOTO 500
300 '
301 ' HAUT
302 '
310 IF Y<1 THEN GOTO 50
320 D$=CHR$(11)+X$ : Y=Y-1 : GOTO 500
400 '
401 ' BAS
402 '
410 IF Y>SY THEN GOTO 50
420 D$=CHR$(10)+X$ : Y=Y+1 : GOTO 500
430 REM AFFICHAGE
500 PRINT CHR$(32);D$;J$;
510 GOTO 50
```

Commentaires

Variables
X$: abrège l'écriture de CHR$(8) un peu long à écrire et qui apparaît plusieurs fois.
SX et SY : limites du déplacement (colonnes et lignes).
X et Y : position courante du caractère. Au début, en haut à gauche.
J$: chaîne formée du caractère plein et d'un retour du curseur sur cette position.
R$: caractère frappé au clavier.
A : code ASCII de R$.
D$: caractère de contrôle pour le déplacement du curseur.

Utilisation d'un chronomètre

Si l'envie vous prend de sophistiquer un programme, vous vous heurterez certainement à un principe bien connu : on ne peut pas faire deux choses à la fois. Le microprocesseur de votre micro-ordinateur n'échappe pas à la règle. Cependant, on peut trouver un terrain d'entente.
Dans l'exemple suivant, nous avons le projet de réaliser un jeu et d'afficher un chronomètre à l'écran. Faire une boucle qui décompte le temps ne pose pas de problème... Oui mais, pendant que le micro-ordinateur égrène les secondes, il ne fait pas jouer son propriétaire.

Programme :
Le chronomètre sera contenu dans un sous-programme.
L'instruction ON INTERVAL... fait référence à l'horloge interne du MO6, celle qui rythme le microprocesseur. Il faut savoir que cette horloge a des intervalles de temps de $1/10^e$ de seconde. C'est pourquoi l'instruction ON INTERVAL = 10 GOSUB provoque un branchement au sous-programme défini par GOSUB toutes les secondes ($10/10^e$ de seconde) et ceci quel que soit l'endroit du programme où l'on se trouve.
Mettre le chronomètre en route consiste alors à passer sur l'instruction INTERVAL ON.

```
1 '
2 ' Jeu
3 '
10 SCREEN 3,0,0 : CLS
20 ON INTERVAL=10 GOSUB 1000
30 INTERVAL ON
40 DRA=0
50 GOSUB 10000
60 IF DRA=0 THEN 50
999 END
1000 '
1001 ' Chronomètre
1002 '
1010 SEC=(SEC+1)MOD 60 : ATTRB 0,0
1020 LOCATE 35,0 : PRINT SEC
1999 RETURN
10000 '
10001 ' Jeu
10002 '
10010 ATTRB 1,1 : COLOR 0,3
10020 LOCATE 10,10 : PRINT CHR$(49+RND*7)
10030 FOR I=1 TO 60
10040 IF INKEY$<>"" THEN DRA=1 : EXIT
10050 NEXT I
10999 RETURN
```

Commentaires :
Le sous-programme de chronomètre, déclenché toutes les secondes, affiche ces secondes en haut à droite de l'écran.
Le sous-programme 10000 affiche un chiffre au hasard au milieu de l'écran à intervalles réguliers. Il s'arrête lorsque vous appuyez sur une touche. Vous constaterez donc, que le programme fait deux choses en même temps. Voici une règle simple pour bien s'amuser : décider que vous devez appuyer lorsque le chiffre 5 sera affiché. Il faut à la fois du réflexe et de la chance !

Module 44 Principes d'animation

Pour la petite et pour la grande informatique, c'est-à-dire pour le "bidouilleur" comme pour l'ingénieur informaticien spécialisé dans la synthèse d'image, les grands principes d'animation sont les mêmes. Pour déplacer un point, un groupe de points, un caractère ou un groupe de caratères, il faut afficher quelque part, afficher un peu plus loin puis effacer l'ancienne position.
Ces déplacements sont la plupart du temps inscrits dans des boucles FOR...NEXT.

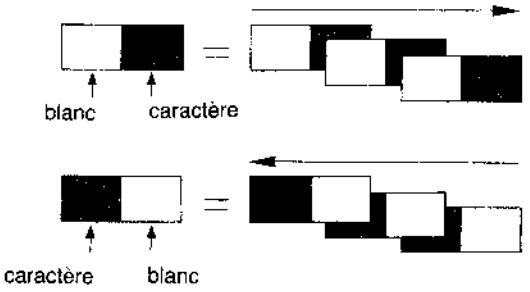
La vitesse apparente du déplacement correspond en fait à la vitesse d'affichage et d'effacement et c'est là où les choses se gâtent en BASIC. Le langage est réputé comme étant particulièrement lent. Il faut bien s'entendre sur ce que signifie cette lenteur. Si vous écrivez une longue boucle d'affichage de nombres, l'écran se remplit très rapidement en général. La lenteur n'apparaît que lorsque les instructions d'affichage sont inscrites dans un programme où les tests et les calculs sont assez importants mais surtout lorsque ce programme est chargé de plusieurs affichages, plusieurs mouvements simultanés. C'est par exemple le cas d'un jeu interactif complexe type guerre des étoiles. La programmation BASIC est tout à fait incompatible avec un écran très mobile : plusieurs envahisseurs, un canon à déplacer, des obus à tester, des bruitages à gérer, etc.

Les concepteurs utilisent le langage machine dont la vitesse d'exécution est plusieurs dizaines de fois supérieure à celle du BASIC. Quoi qu'il en soit, notre brave BASIC est capable de rendre de bons services pour des animations simples. Il arrive même lorsque ces animations sont très simples que la vitesse soit si grande que les yeux ont de la peine à suivre le mouvement.

Oscillations horizontales

Le but de ce programme est de faire osciller un caractère sur une ligne horizontale.

Pour le mouvement de gauche à droite, on affiche le caractère précédé d'un blanc qui efface la position précédente.
Pour le mouvement de droite à gauche, on affiche le caractère suivi d'un blanc qui efface la position précédente.



Pour les déplacements de gauche à droite, on doit faire revenir le curseur d'une colonne à gauche et de trois colonnes pour les déplacements de droite à gauche (X$ et XX$).
C'est le même sous-programme qui gère les deux déplacements.
Les variables d'entrée sont :
— A et B les numéros de colonnes extrêmes.
— D$ le caractère de retour de curseur.
— P l e pas.
— C$ la chaîne à imprimer.

```
1 '
2 ' OSCILLATIONS
3 '
20 CLS
30 LOCATE 0,10,0
40 X$=CHR$(8) : XX$=X$+X$+X$
45 DO
```

```
50 REM GAUCHE DROITE
60 A=0 : B=20 : D$=X$ : P=1 : C$=" *"
70 GOSUB 1000
80 REM DROITE GAUCHE
90 A=20 : B=0 : D$=XX$ : P=-1 : C$="* "
100 GOSUB 1000
110 LOOP
120 END
1000 '
1001 ' DEPLACEMENT
1002 '
1010 FOR K=A TO B STEP P
1020 PRINT C$ ; D$ ;
1030 FOR N=1 TO 10 : NEXT N
1040 NEXT K
1099 RETURN
```

Commentaires :
A vous de choisir le test de sortie de boucle DO...LOOP.
On a placé dans la boucle d'affichage une autre boucle qui sert de temporisation pour freiner le déplacement.

Utile : tous les quadrillages

Plusieurs années d'observation des programmes sur micro-ordinateur amènent à la conclusion suivante : il est extrêmement rare de trouver un programme de jeu ou un programme éducatif dans lequel il n'y ait pas au moins un quadrillage...
On propose ici un sous-programme universel qui dessine n'importe quel quadrillage.

Programme :
Variables principales du programme caractérisant un quadrillage :
— Coordonnées du coin supérieur gauche : SX et SY
— Nombre de colonnes : NC
— Longueur d'une colonne : LC
— Hauteur d'une ligne : HL
— Couleur : C

```
1 '
2 ' Tous les quadrillages
3 '
10 CLS : SCREEN 3,0,0
20 SX=50 : SY=20
30 NC=10 : NL=8          } Caractéristiques
40 LC=20 : HL=12         } du quadrillage
50 C=3
60 GOSUB 1000
999 END
1000 '
1001 ' Quadrillage
1002 '
1010 TX=SX+NC*LC : TY=SY+NL*HL
1020 FOR X=SX TO TX STEP LC
1030 LINE (X,SY)-(X,TY),C
1040 NEXT X
1050 FOR Y=SY TO TY STEP HL
1060 LINE (SX,Y)-(TX,Y),C
1070 NEXT Y
1999 RETURN
```

Commentaires : pour utiliser le sous-programme, il suffit d'agir sur les 7 variables d'entrée définies dans notre exemple entre les lignes 20 et 50.
— TX et TY sont les coordonnées du coin inférieur droit.

Module 45 Derniers conseils de programmation

Ecrire de petits programmes ne pose pas de problème important. On retrouve toujours plus ou moins facilement ce qu'on a voulu faire, les astuces utilisées et pourquoi telle instruction a cette forme étrange. Pour des programmes plus importants, on ne peut plus faire confiance à la mémoire du programmeur pour se souvenir, quelques mois après, de la réalisation des procédés spécifiques utilisés. A ce sujet, le BASIC est un langage dangereux. Il permet une programmation immédiate et n'oblige pas le programmeur à structurer ses programmes.
La méthode de programmation proposée au module 22 vous incitait à décomposer les programmes en sous-tâches élémentaires. Avec une telle méthode, on peut penser que les programmes résultants seront clairs et faciles à relire.
De toute façon, la structuration d'un programme n'est pas contenue dans le langage mais dans la tête du programmeur !
Les quelques conseils qui suivent précisent certains points délicats : les sorties de boucles.

FOR...NEXT et DO...LOOP

Pour entrer dans une boucle FOR...NEXT, il faut impérativement entrer dans la boucle par la ligne où figure FOR et en sortir par la ligne où figure NEXT.
De même, il est interdit d'entrer dans une boucle DO...LOOP ailleurs que par le DO.
L'ignorance ou l'oubli de ces deux principes est la source de nombreux ennuis. Il ne faut jamais l'oublier car les instructions GOTO vous permettent de rentrer et de sortir des boucles dans le plus grand désordre.
D'une façon précise, si une boucle est installée entre les lignes 100 et 150, aucun GOTO extérieur à la boucle ne devra envoyer sur un numéro de ligne compris entre 100 et 150. De même, entre les lignes 100 et 150, aucun GOTO ne devra envoyer sur un numéro de ligne inférieur à 100 ou supérieur à 150. En revanche, les GOSUB sont autorisés dans la boucle car le retour est assuré.
Aucune exception n'est faite à l'entrée dans la boucle. En revanche, on peut sortir d'une boucle sans la dérouler entièrement.
Pour sortir d'une boucle FOR...NEXT, aussi bien que d'une boucle DO...LOOP, sans la dérouler entièrement, le mieux est d'utiliser l'instruction EXIT. Tout autre méthode (GOTO notamment) doit être évitée car elle fait perdre l'avantage d'un programme bien structuré.
Dans le cas de boucles emboîtées, il est conseillé d'avoir une ligne d'instruction par fin de boucle même si le langage vous autorise à terminer plusieurs boucles sur la même ligne. La lisibilité du programme est ainsi bien meilleure.
En cas de grande confusion dans le déroulement d'un programme, il est possible de demander l'avis du micro-ordinateur lui-même. L'instruction TRON (pour TRACE ON) remet en général les idées en place.

TRON-TROFF

L'ordre TRON permet de visualiser, lors de l'exécution d'un programme, les numéros des instructions effectuées. Les numéros d'instruction sont affichés entre crochets pour que l'utilisateur s'y retrouve.

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

La commande TROFF annule la commande TRON.
Puisque nous sommes intéressés par les sorties de boucles, il faut en profiter pour signaler quelques ennuis qui vous guettent.
L'ennemi du programmeur est le "programme qui boucle", c'est-à-dire le programme qui n'arrive pas jusqu'à l'instruction END.

Un cas fréquent de bouclage se produit dans les tests concernant une précision numérique.
Nous attirons donc votre attention sur le fait que les chiffres utilisés par le micro-ordinateur sont codés et que la précision du codage fait parfois apparaître quelques surprises.

Exemple :
Le programmeur dispose d'une notion de zéro et la machine a deux représentations du zéro : une pour les nombres entiers (codée sur deux octets) et une pour les nombres réels. Tester si un nombre réel est nul en écrivant IF X=0 peut alors conduire à des problèmes. On préférera l'écriture IF ABS(X)<1.E-7.
La précision du codage étant de 7 chiffres significatifs, ce test est suffisant. Il implique que 0,0000001 et 0 sont les deux mêmes nombres pour le micro-ordinateur.
Dans le même ordre d'idées, nous allons écrire un petit programme qui manipule des grands nombres. Nous vous rappelons que la limite des nombres utilisables en simple précision est de 10^{+38}.
Le dépassement de cette valeur correspond à une erreur *"overflow"*.

Exemple :
Nous voulons calculer :

$$\frac{A^n}{n!} = \frac{A \times A \times \ldots \times A}{1 \times 2 \times 3 \times \ldots \times n}$$

La première idée consistant à faire une boucle pour calculer le numérateur et une seconde évaluant le dénominateur conduira à un dépassement de capacité dès que n sera grand. Il faut donc calculer le rapport des deux expressions au fur et à mesure qu'on les évalue de façon à rester dans les limites admises.

Programme :

```
10 REM GRANDS NOMBRES
20 N=100 : A=50
30 S=A
40 FOR I=2 TO N
50 S=S*A/I
60 NEXT I
99 END
```

Commentaire : à vous de programmer l'autre méthode et de comparer les résultats.

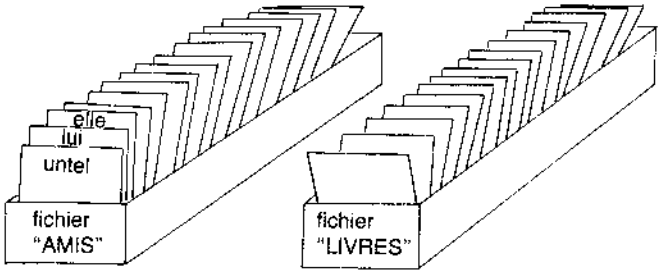
Remarque : toutes les erreurs apparaissant dans les programmes peuvent être gérées par le programmeur. Il suffit d'utiliser les instructions ON ERROR... et RESUME (voir la partie "références").

Module 46 Les fichiers séquentiels

Au chapitre 20 nous avons montré comment on peut enregistrer un programme sur une cassette et le réutiliser ensuite. Sur la cassette, on dit qu'on a constitué un fichier-programme.

Qu'est-ce qu'un fichier ?

Un "fichier" est une suite de données rangées les unes derrière les autres comme des "fiches" dans une boîte.



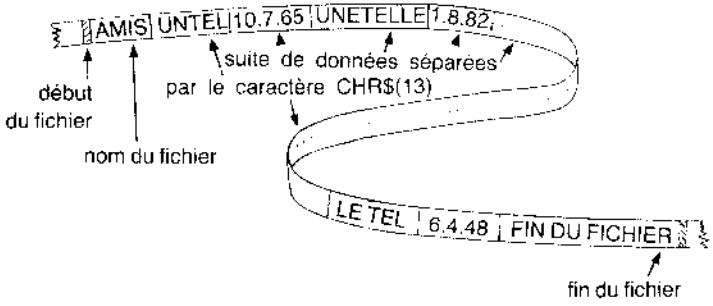
Chacune des "fiches" contient un certain nombre d'informations relatives à "l'enregistrement" correspondant.
Dans le cas d'un fichier-programme, chaque fiche contient une instruction numérotée du programme qui est enregistré.
Il existe un second type de fichier : les fichiers de données. A la place des instructions numérotées sont enregistrées des données, c'est-à-dire des listes de mots comprenant des chaînes de caractères et/ou des nombres.
Il peut s'agir, par exemple, du fichier de vos amis, chaque fiche contenant son adresse, son numéro de téléphone et sa date de naissance ; ou bien du fichier des clients d'une entreprise, chaque fiche contenant ses caractéristiques, l'état de ses commandes et de son compte ; ou bien du fichier des livres d'une bibliothèque...
Stocker ses données dans un fichier plutôt que dans un programme présente plusieurs avantages : d'abord on peut stocker beaucoup plus de données sur cassette, et ensuite un même fichier de données peut servir dans plusieurs programmes différents.
Notons d'abord qu'il est impossible d'enregistrer, de charger et encore moins d'exécuter un fichier de données avec une seule instruction comme on le fait avec un fichier-programme (SAVE, LOAD, RUN).

Gestion d'un fichier

Cet ensemble de fiches est destiné à être ou bien feuilleté, ou bien modifié (addition ou soustraction d'une fiche). Pour consulter un fichier ou pour écrire dedans, il faut successivement :
— Trouver la bonne boîte (grâce à son "nom", par exemple) et "l'ouvrir".
— Feuilleter les fiches jusqu'à ce que l'on trouve celle qui nous intéresse.
— Prendre la fiche et l'amener devant celui qui doit la traiter (par le "canal" approprié)
— Traiter les informations contenues dans la fiche ou la modifier ; puis la remettre dans le fichier.
— Ne pas oublier de "fermer" la boîte (sinon les couvercles se mélangent).

Fichiers sur cassettes

Pratiquement un fichier sur cassette est constitué :
— d'un premier enregistrement contenant le nom du fichier,
— d'une liste de données alphabétiques ou numériques,
— d'un code signalant la fin des enregistrements de ce fichier.



On dit que l'accès à ce type de fichier est séquentiel (d'où leur nom). Cela signifie qu'il n'est pas possible d'atteindre une donnée quelque part au milieu du fichier sans avoir fait défiler la suite de toutes les données précédentes. En effet, aucune des données n'a de "nom" permettant de l'"appeler" ; pour consulter un tel fichier il faut avoir bien noté la structure des enregistrements au moment de sa constitution afin de "reconnaître" chacune des données au fur et à mesure de leur défilement. Dans l'exemple du schéma précédent, chaque enregistrement du fichier AMIS est constitué d'un nom et d'une suite de chiffres représentant la date de naissance.

Ecrire un enregistrement

Un fichier de données est stocké sur une cassette, c'est-à-dire sur une mémoire de masse. Le micro-ordinateur travaille, lui, uniquement avec la mémoire centrale (MEM et MEV). Ecrire un enregistrement dans un fichier consiste donc à transférer des données d'une mémoire vers une autre.
Pour cela, il faut d'abord choisir un numéro de canal.
En effet les données situées en mémoire centrale passent d'abord par une sorte de sas ou "zone tampon" (en anglais *buffer)* avant d'être transférées sur la cassette.

OPEN, WRITE# et CLOSE

L'instruction

OPEN "0",#1,"AMIS"

ouvre le fichier de nom "AMIS" en Output (c'est-à-dire que l'on va écrire dessus) ; le canal réservé à sa manipulation par le MO6 porte le numéro 1.

Exemple : on désire constituer un fichier de données comprenant les noms et numéros de téléphone de quelques amis.
Les informations à enregistrer sont entrées par dialogue au clavier.

```
5 CLS
10 'ECRITURE ANNUAIRE
20 OPEN "O",#1,"ANNUAIRE"
40 DO
50 INPUT "NOM :" ; NOM$
60 IF NOM$="" THEN EXIT
70 INPUT "TELEPHONE",TEL$
80 WRITE#1,NOM$,TEL$
90 LOOP
100 CLOSE#1
110 PRINT"ECRITURE TERMINEE"
999 END
```

Commentaires :
les lignes 50 à 80 constituent une boucle d'écriture des enregistrements. La boucle est terminée lorsqu'on entre un nom vide, c'est-à-dire lorsqu'on appuie directement sur ENTRÉE en réponse à la question posée ligne 50.
100 : Ne pas oublier de fermer le canal après son utilisation. L'instruction CLOSE#1 commande cette fermeture.
Evidemment, avant d'exécuter ce programme, il faut que le LEP soit en position d'enregistrement avec une bande magnétique dans le lecteur. Vous devez avoir bien repéré le début de l'enregistrement qui va avoir lieu.

Module 47 Ecriture/Lecture

PRINT#

Voici un autre exemple d'enregistrement de données dans un fichier sur cassette. On y voit qu'une suite d'instructions "DATA", dans un programme, n'est pas autre chose qu'un fichier séquentiel. Ce fichier est consulté grâce à l'instruction "READ". Le "pointeur-de-DATA" joue le même rôle que la tête de lecture du LEP : à un instant donné, il est prêt à lire la donnée suivante dès que l'ordre lui en est donné. Remarquons encore que DATA signifie une donnée en anglais.

Programme :

```
10 DATA UNTEL,345126 33,LUI,235426 47,
MOI,4010 37 48
11 DATA ELLE,720914 10,PERSONNE,456678 90
19 DATA Z,0
100 '
101 ' ENREGISTREMENT
102 '
110 OPEN"O",#1,"TELEPHON"
115 DO
120 READ A$,T$
130 IF A$="Z" THEN EXIT
140 PRINT #1,A$,T$
150 LOOP
190 CLOSE#1
199 END
```

Commentaires :

— La donnée de l'instruction 19 permet de repérer la fin de la liste des informations à enregistrer.

— L'instruction :

```
PRINT#1,A$,T$
```

commande l'envoi des variables A$ et T$ par le canal 1 (lequel a été affecté à un certain nom de fichier par l'instruction OPEN).

Note : l'instruction PRINT# enregistre les données sous la même forme que l'instruction PRINT les affiche à l'écran.

Lire un enregistrement

Pour lire dans le fichier que nous venons d'enregistrer, il faut :

— S'assurer que le lecteur-enregistreur est en position de lecture avec la cassette positionnée à lire avant l'endroit où est enregistré le fichier.

— Ouvrir la communication entre la mémoire centrale et la cassette par l'intermédiaire d'un canal qui n'est pas nécessairement le même que celui de l'écriture.

— Transférer en mémoire centrale le contenu du fichier et éventuellement l'afficher à l'écran.

— Fermer la communication entre la mémoire centrale et la cassette.

Exemple :

```
1 REM *** LECTURE ***
10 CLEAR 600
20 DIM A$(30),T$(30)
100 OPEN "I",#1,"TELEPHON"
110 I=1
120 DO
130 IF EOF(1) THEN EXIT
140 INPUT #1,A$(I),T$(I)
150 I=I+1
160 LOOP
200 CLOSE #1
999 END
```

Commentaires :

— L'instruction

```
OPEN"I",#1,"TELEPHON"
```

ouvre le fichier "TELEPHON" (c'est-à-dire que l'on va lire dessus) ; le canal réservé à sa manipulation porte le numéro 1.

— La lecture des données se fait par l'intermédiaire de la commande INPUT#. Elle s'emploie comme l'instruction INPUT quant à la syntaxe.

Au fur et à mesure, les données se voient nommées et rangées comme les éléments de deux tableaux, ce qui permettra ensuite de les rappeler (l'instruction 1 réserve trente places en mémoire pour chacun de ces tableaux).

EOF

Dans l'exemple précédent, le test de sortie de boucle introduit une variable EOF(1).
Le 1 signifie qu'elle se rapporte au canal n°1.
EOF est l'abréviation de *end of file* qui signifie fin de fichier. En effet, quand on lit dans un fichier, il est important de tester après chaque lecture si la fin du fichier a été rencontrée, sinon, vous obtiendrez le message d'erreur suivant :
"Input Past End" c'est-à-dire lecture après la fin du fichier.
La variable EOF(1) est du type logique. Elle prend la valeur VRAI lorsque la dernière donnée du fichier traité sur le canal 1 vient d'être lue. Le test est donc un test logique (et on n'écrit pas : IF EOF(1)=0...).

Remarque : comme pour les noms des fichiers-programmes, les noms des fichiers de données comportent au maximum huit caractères sans virgule ni point.

Recensement général des périphériques utilisables avec un fichier

Sans autre indication, l'ouverture du canal n°1 concerne la cassette. Si vous utilisez simultanément un lecteur de disquettes, celui-ci deviendra prioritaire sur le lecteur de cassettes. Voici les différents périphériques que vous pouvez utiliser.

en écriture

OPEN"O",#1,"CASS:ANNUAIRE"
(CASS est facultatif en l'absence d'un lecteur de disquettes)
pour écrire sur la cassette,

OPEN"O",#1,"LPRT:"
pour écrire sur imprimante,

OPEN"O",#1,"SCRN:"
pour écrire à l'écran,

OPEN"O",#1,"COMM:"
pour écrire sur la voie de communication série.

L'ensemble "CASS:ANNUAIRE" constitue le descripteur de fichier. Pour les trois derniers périphériques, il est inutile de préciser le nom du fichier puisqu'il est seulement transmis et non pas conservé dans une mémoire.

en lecture

OPEN"I",#1,"1:ANNUAIRE"
pour lire à partir du lecteur n°1,

OPEN"I",#1,"CASS:ANNUAIRE"
pour lire à partir de la cassette,

OPEN"I",#1,"KYBD:"
pour lire à partir du clavier (KEYBOARD)

OPEN"I",#1,"COMM:"
pour lire à partir de la voie série.

Il est évidemment impossible de "lire" un fichier écrit à l'écran ou sur l'imprimante : ces deux périphériques sont exclusivement des périphériques "de sortie".
Seuls les lecteurs de disquettes, de cassettes et la voie de communication série peuvent être utilisés aussi bien en entrée qu'en sortie.
Deux périphériques sont un peu différents des autres : l'écran et le clavier. En effet, les deux instructions du BASIC standard INPUT et PRINT permettent à l'ordinateur de "lire" à partir du clavier (entrée des données) ou "d'écrire" à l'écran (affichage) sans utiliser de canal.
C'est pour mettre ces deux périphériques en parallèle avec les autres que vous est offerte la possibilité d'ouvrir un fichier en lecture avec le clavier (KYBD:) et en écriture avec l'écran (SCRN:).

Module 48 Manipulations de fichiers

Exemple de sortie sur imprimante

```
10 REM * LECTURE * ECRITURE *
20 CLEAR 600
30 DIM A$(30),T$(39)
100 OPEN"I",#1,"TELEPHON"
110 GOSUB 1000
120 CLOSE #1
200 OPEN"O",#2,"LPRT"
210 GOSUB 2000
220 CLOSE #2
999 END
1000 REM LECTURE
1010 I=1
1020 DO
1030 IF EOF(1) THEN EXIT
1040 INPUT #1,A$(I),T$(I)
1050 I=I+1
1060 LOOP
1999 RETURN
2000 REM ECRITURE
2010 FOR N=1 TO I
2020 PRINT #2,"NOM : " ; A$(N), "TELEPHONE : " ; T$(N)
2030 NEXT N
2040 RETURN
```

Commentaires :
Le sous-programme 1000 est la copie conforme de celui décrit pour lire des enregistrrements. Il fournit en plus le nombre I d'enregistrements lus. Ce nombre est inférieur à 30, dimension prévue pour recevoir la lecture.
Le sous-programme 2000 écrit sur l'imprimante. La boucle utilise le nombre I donné par le premier sous-programme.

Variations :
Il est possible de ne faire qu'une seule boucle en lisant un enregistrement sur le canal 1 et en le réécrivant immédiatement sur le canal 2. On s'affranchit ainsi des deux tableaux définis en début de programme. Cependant, il faut bien remarquer que cette méthode ne permet pas de traiter ultérieurement les enregistrements lus puisqu'ils viennent régulièrement s'écraser les uns les autres.

Remarque : pour une meilleure présentation sur imprimante, on peut utiliser l'instruction PRINT#USING qui fonctionne comme PRINT USING.

Un fichier de nombres : les points d'un dessin

Les coordonnées des points d'un dessin peuvent être placées dans une instruction DATA. Mais en les mettant dans un fichier, on peut les appeler dans n'importe quel programme, et utiliser ce dessin sans avoir à réécrire la liste des coordonnées.

Exemple :

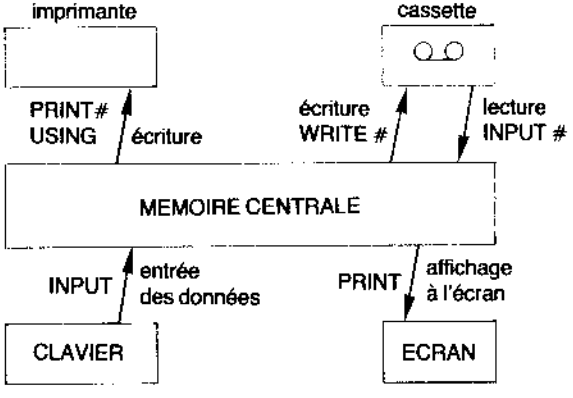
```
10 * DESSIN *
20 OPEN"O",#1,"DESSIN"
30 DO
40 INPUT"COL,LIG" ; COL,LIG
50 IF COL=-1 THEN EXIT
60 WRITE#1,COL,LIG
70 LOOP
80 CLOSE#1
99 END
```

Commentaires :
Le test de sortie de boucle utilise la valeur −1 pour l'indice des colonnes (et n'importe quelle valeur pour LIG).
Ce programme vous oblige à entrer les coordonnées une par une. Nul doute que vous trouverez une méthode moins pénible pour arriver au même résultat.
Vous pouvez par exemple entrer votre dessin au crayon optique en remplaçant le INPUT de la ligne 40 par un INPUTPEN. Mais attention, le caractère séquentiel du fichier vous interdit tout remords et toute correction.
L'utilisation des points mémorisés peut se faire de la manière suivante :

Programme :

```
10 ' DEMONSTRATION
20 CLS
30 OPEN"I",#1,"DESSIN"
40 INPUT#1,C,L
50 PSET(C,L)
60 DO
70 IF EOF(1) THEN EXIT
80 INPUT#1,COL,LIG
90 LINE−(COL,LIG)
100 LOOP
110 CLOSE
999 END
```

Commentaires :
— Le premier point lu servira de point de départ au dessin.
— Le programme utilise directement les valeurs contenues dans le fichier sans les afficher.
Résumons les différentes possibilités offertes par les fichiers séquentiels et que nous avons présentées ici :



MERGE

Terminons en signalant qu'il existe des opérations portant sur les fichiers dont une est particulièrement importante quand elle concerne les fichiers-programmes.
L'instruction MERGE (anglais de fusion) permet de charger un programme enregistré sur une mémoire de masse (ici la cassette) dans la mémoire centrale du micro-ordinateur et ceci sans effacer le programme qui s'y trouve déjà.
Le but est d'enchaîner les deux programmes et de les faire s'exécuter à l'aide d'une seule instruction RUN.
La première précaution à prendre concerne les numéros des lignes d'instructions : il ne faut pas que le même numéro de ligne apparaisse dans chacun des deux programmes sous peine de tout brouiller.
La seconde précaution concerne le type de sauvegarde. Le programme que vous appellerez de la cassette doit y être enregistré sous forme de fichier ASCII.
Pour ce faire, il faut ajouter le suffixe A à l'aide de SAVE :

```
SAVE "TRUC",A
```

réalise une sauvegarde ASCII.

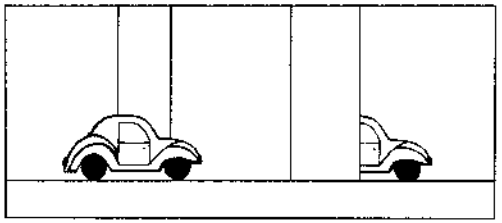
Module 49 L'effet caméléon

Consolez-vous, voici l'effet caméléon

Ce titre énigmatique fait référence à l'instruction CONSOLE grâce à laquelle on peut réaliser un effet qui rappelle la vertu du caméléon de prendre la couleur des endroits où il séjourne.
CONSOLE peut être suivi de quatre arguments.
L'effet caméléon concerne le troisième argument.
Les deux premiers arguments définissent une fenêtre de défilement (en anglais SCROLLING) ; cette instruction est l'équivalent horizontal de l'instruction WINDOW.
Le quatrième argument contrôle la vitesse de défilement :
0 correspond à la vitesse normale, c'est-à-dire assez rapide,
1 correspond à une vitesse beaucoup plus lente, un défilement doux.

Le troisième argument de console : l'effet caméléon

Ce troisième argument permet des effets d'animation très surprenants. Pour en profiter, il faut lui donner la valeur 1. Ainsi CONSOLE 0, 24, 1, 0, permettra d'utiliser l'effet caméléon sur tout l'écran.
Le principe est le suivant : au moment où cette instruction apparaît, toutes les couleurs présentes sur l'écran sont figées jusqu'à une nouvelle instruction CONSOLE. Il est possible d'imprimer et de déplacer des caractères sur tout l'écran mais, à chaque emplacement où un caractère sera imprimé, les couleurs de forme et de fond seront celles fixées avant l'instruction CONSOLE.
Lorsque tout l'écran est sous effet caméléon, il est inutile d'utiliser l'instruction COLOR. Avant de passer sous ce mode, le programmeur devra donc prendre le soin de prévoir exactement ce qui adviendra par la suite.
Voici une petite auto rouge qui roule sur une route bordée de hauts platanes. Vus du bas-côté, les arbres du premier plan paraissent plus larges que ceux de l'arrière-plan. Lorsque la voiture passe derrière un arbre du premier plan, elle disparaît, alors que lorsqu'elle passe devant un arbre de l'arrière-plan, elle reste visible.



Pour réaliser cette scène saisissante qui donnera à peu de frais une impression de profondeur, il faut commencer par construire une auto grâce à deux caractères.
Pour le reste, on fixera dans un premier temps une couleur de forme et une couleur de fond sur chaque partie du paysage concernée par le déplacement de l'auto : le ciel et les arbres.
Les arbres sont dessinés avec des caractères pleins.
Après l'instruction CONSOLE 0,24,1, tout caractère imprimé sur le ciel apparaîtra en rouge ; tout caractère imprimé sur un gros arbre apparaîtra en orange sur fond orange, donc n'apparaîtra pas ; tout caractère imprimé sur un arbre fin apparaîtra en rouge sur fond orange.

Programme :

```
1 '
2 ' AUTO
3 '
20 CLEAR,,2
30 DEFGR$(0)=15,25,49,127,255,255,60,24
40 DEFGR$(1)=224,48,16,254,255,255,30,12
50 AU$=GR$(0)+GR$(1)
60 ' PAYSAGE
70 SCREEN 1,12,0 : CLS
80 ATTRB 1,1 : LOCATE 0,0,0
90 BOXF(0,120)-(319,199),2
100 ' ARBRES
```

```
110 FOR X=6 TO 38 STEP 8
120 BOXF(X,0)-(X+1,14)" ",15,15
130 NEXT X
140 FOR X=2 TO 34 STEP 8
150 BOXF(X,0)-(X,14)" ",1,15
160 NEXT X
170 ' CAMELEON
180 CONSOLE 0,24,1
190 X=0
200 LOCATE X,14 : PRINT " "+AU$+" "
210 DO
220 R$=INPUT$(1) : R=ASC(R$)
230 IF R=13 THEN EXIT
240 IF R=8 AND X>0 THEN X=X-1 ELSE IF R=9 AND X<33 THEN X=X+1
ELSE GOTO 220
245 LOCATE X,14 : PRINT"$"+AU$+"$"
250 LOOP
260 ATTRB 0,0 : LOCATE 0,0 : CONSOLE 0,24,0
999 END
```

Commentaires :
20-50 Définition de l'auto, AU$ à partir des deux caractères GR$(0) et GR$(1).
70 Définition de la couleur du ciel : rouge sur fond bleu pastel.
90 Dessin de la prairie (sans importance).
110-130 Dessin des arbres du premier plan. Boîtes pleines de caractères « espace » orange sur fond orange.
140-160 Dessin des arbres du deuxième plan. Boîtes pleines de caractères « espace » rouges sur fond orange.
Tous les arbres apparaissent orange.
180 Voici l'effet caméléon.
200 Affichage de l'auto à gauche.
210 Enregistrement du code ASCII de la touche frappée (voir Interactivité).
220 L'appui sur ENTREE signale la fin du divertissement et le retour aux conditions normales.
230 Marche arrière (premier test)
Marche avant (second test)
et détection d'une mauvaise touche frappée

Mode d'emploi
1 — Entrer le programme puis faire RUN.
2 — L'auto se déplace grâce aux touches ← et →.
3 — ENTREE permet d'arrêter le déroulement du programme.

Remarque : à la ligne 240, on a un exemple de deux tests emboîtés IF...THEN...ELSE IF...THEN...ELSE...
On peut aussi bien emboîter dans une partie THEN que dans une partie ELSE.
Il faut se méfier des niveaux d'emboîtement pour ne pas être surpris lors de l'exécution !
En cas d'ennui, si le programme s'interrompt avant de passer en ligne 999, vous devrez remettre les couleurs de l'écran dans leur état initial. Tapez donc, en mode direct, la commande

```
CONSOLE 0,24,0
```

Module 50 Rock and Stock

Le rock and roll est construit à partir de trois accords appartenant à une catégorie qu'on appelle "Majeur septième". L'important dans un accord n'est pas seulement la hauteur des notes que l'on joue mais les intervalles qui les séparent. Ainsi, un accord de "Majeur septième" peut être représenté par les quatre nombres 0, 4, 7, 10 signalant les écarts, comptés en demi-tons, entre les notes qui le composent.

Voici par exemple l'accord DO Majeur septième que l'on note souvent DO7 :

Do	Do#	Ré	Ré#	Mi	Fa	Fa#	Sol	Sol#	La	La#	Si
0	1	2	3	4	5	6	7	8	9	10	11

Un rock and roll est classiquement une suite de douze mesures, correspondant chacune à un accord. La voici :

Do7	Do7	Do7	Do7	Fa7	Fa7	Do7	Do7	Sol7	Fa7	Do7	Do7

Chaque mesure est composée de huit croches.
Pour construire chacune de ces mesures, il suffira de choisir au hasard huit notes dans l'accord correspondant.
Notons enfin que si l'on prend la hauteur du premier accord comme base, la suite des douze accords peut, elle aussi, être codée par une suite de douze nombres.

Do7	Do7	Do7	Do7	Fa7	Fa7	Do7	Do7	Sol7	Fa7	Do7	Do7
0	0	0	0	5	5	0	0	7	5	0	0

Pour que l'air soit vraiment réussi, il faudra que la suite de notes jouées soit la même pour chaque accord. Supposons par exemple que le hasard donne la suite
4, 4, 7, 0, 0, 10, 7
pour les huit notes de l'accord de Do7, ce sont ces mêmes huit intervalles que nous reprendrons pour tous les autres accords du morceau.

Le programme :

```
1 '
2 ' Random Rock
3 '
10 DATA DO,DO#,RE,RE#,MI,FA,FA#,SO,SO#,LA,LA#,SI
20 DIM NT$(60)
30 GOSUB 1000
40 '
50 DATA 0,4,7,10
60 DATA 0,0,0,0,5,5,0,0,7,5,0,0,99
70 GOSUB 2000:GOSUB 3000
80 PLAY "L12":RESTORE 50
90 FOR K=1 TO 4:READ HN(K):NEXT K
100 GOSUB 4000
110 RESTORE 60
120 READ ACC
130 IF ACC=99 THEN 110
140 FOR X=1 TO 8
150 PLAY NT$(TN+ACC+SM(X))
160 NEXT X
170 GOTO 120
999 END
1000 '
1001 ' Construction du tableau
1002 '
1010 FOR J=0 TO 4
1020 O$="O"+CHR$(49+J)
1030 RESTORE 10
1040 FOR K=1 TO 12
1050 READ T$: NT$(12*J+K)=O$+T$
1060 NEXT K
1070 NEXT J
1999 RETURN
2000 '
2001 ' Mélange
```

```
2002 '
2010 PRINT "APPUYEZ SUR UNE TOUCHE"
2020 Z=RND
2030 IF INKEY$="" THEN 2020
2999 RETURN
3000 '
3001 ' Ton
3002 '
3010 PRINT"CHOISISSEZ LE TON ENTRE 0 ET 43:":
3030 INPUT TN
3999 RETURN
4000 '
4001 ' Suite mélodique
4002 '
4010 FOR X=1 TO 8
4020 K=1+INT(RND)
4030 SM(X)=HN(K)
4040 NEXT X
4999 RETURN
```

Commentaires :

— Le sous-programme 1000 construit un tableau qui contient les 60 notes "jouables" par le MO6 :

Octave												
	1	DO	DO#	RE	RE#	MI	FA	FA#	SO	SO#	LA	LA#	SI
	2	DO	DO#	RE	RE#	MI	FA	FA#	SO	SO#	LA	LA#	SI
	3	DO	DO#	RE	RE#	MI	FA	FA#	SO	SO#	LA	LA#	SI
	4	DO	DO#	RE	RE#	MI	FA	FA#	SO	SO#	LA	LA#	SI
	5	DO	DO#	RE	RE#	MI	FA	FA#	SO	SO#	LA	LA#	SI

— Le sous-programme 2000 "initialise le hasard".

Le programme principal commence par les lignes de DATA 50 et 60 où vous reconnaîtrez facilement les listes de nombres que nous avons construites précédemment. La suite en ligne 60 se termine par le nombre 99 qui indique la fin du morceau.

Le sous-programme 3000 appelé en ligne 70, après le sous-programme 2000, vous permet de choisir le ton du morceau. Il ne faut pas dépasser 43 car dans l'accord le plus haut du morceau (Sol7), la note la plus aiguë se trouve à 17 intervalles de la note la plus basse, prise comme point de départ : 43 = 60 – 17.

La ligne 80 fixe la longueur de toutes les notes (croches). Vous pouvez la modifier à votre guise pour accélérer ou ralentir l'exécution.

La ligne 100 envoie au sous-programme 4000 qui construit la suite mélodique du morceau. Le tableau SM(X) contient huit nombres choisis au hasard parmi 0, 4, 7 et 10.

La boucle 140-160 est le cœur du programme ; c'est elle qui est chargée d'exécuter le morceau. Elle correspond aux huit notes de chaque accord.
Le numéro de la note du tableau NT$ est calculé à partir :
— du ton (TN) choisi par l'utilisateur dans le sous-programme 3000 ;
— du numéro de l'accord (ACC) lu en ligne 120 ;
— du numéro de la Xème note de la suite mélodique.

Tout cela est un peu compliqué ?
Exact ! Donnez-vous un peu de mal pour essayer de bien comprendre, vous n'en apprécierez que mieux le résultat.

Variations

Vous pouvez vous permettre de changer de nombreuses données dans ce programme pour obtenir encore plus de mélodies.
Le plus amusant à notre avis consiste à changer les données des accords.
On obtient par exemple un rock triste en remplaçant l'accord majeur septième par un accord mineur qui est un type d'accord plutôt utilisé pour les airs lents et romantiques.
L'accord mineur est obtenu avec les notes 0, 3, 7, 15. Il vous suffit donc, sans rien modifier aux autres lignes du programme, de taper :

```
50 DATA 0, 3, 7, 15
```

Annexe

Voici deux programmes BASIC vous permettant de sauvegarder et de relire votre palette de couleurs sur votre lecteur de cassettes.

Pour cela, tapez sous BASIC le programme suivant:

```
10 ' Lecture du fichier palette
20 '
30 NOM$="CASS : PALETTE.CFG"
40 '
50 OPN "I", #1, NOMS$
60 FOR COULEUR=0 TO 15
70 PALETTE COULEUR, CVI(INPUT$(2,#1))
80 NEXT
90 CLOSE
100 END
```

et sauvegardez-le par SAVE"LITPAL.

Tapez ensuite le programme suivant:

```
10 ' Ecriture du fichier palette
20 '
30 FOR COULEUR=0 TO 15
40 A$=A$+MKI$(PALETTE(COULEJR))
50 NEXT COULEUR
60 '
70 OPEN "O", #1, "CASS : PALETTE.CFG
80 PRINT #1, A$
90 CLOSE #1
100 END
```

et sauvegardez-le par SAVE"SAUVEPAL.

Pour sauvegarder votre palette, vous rembobinez la cassette et vous tapez: RUN"SAUVEPAL. Votre palette va être sauvegardée dans un fichier dont le nom est PALETTE.CFG. Pour relire votre palette, vous rembobinez la cassette et vous tapez RUN"LITPAL.