QL

Guide de l’utilisateur

Guide du débutant

TABLE DES MATIERES

CHAPITRE 1

PRESENTATION DU QL

L'ECRAN

Votre QL doit être connecté à un moniteur ou à un téléviseur, et cet appareil doit être mis en marche. Appuyer sur quelques touches, et presque aussitôt, l'écran vous apparaîtra comme il est indiqué ci-dessous. La petite lumière clignotante est appelée « curseur ».

Moniteur

Téléviseur

Si votre écran ne vous apparaît pas comme ceci, lisez la partie : « introduction » ; cela pourrait vous aider à résoudre vos problèmes.

LE CLAVIER

Le QL est un ordinateur puissant et aux fonctions variées, aussi n'aurez-vous pas besoin tout de suite de certaines caractéristiques du clavier. Pour l'instant, nous n'allons présenter que les éléments dont vous aurez besoin pour ce chapitre et les six suivants.

BREAK

Ceci vous permet de sortir de situations embarrassantes. Par exemple :

- une ligne que vous avez décidée d'abandonner,
- quelque chose de faux que vous n'avez pas compris,
- un programme en cours qui n'a plus d'intérêt,
- n'importe quelle autre problème…

BREAK est si puissant qu'il ne doit pas être frappé accidentellement. Pour l'effectuer, appuyer sur **CTRL** puis sur **SPACE**.

La séquence **BREAK** peut être utilisée pour stopper un programme en cours, parce que quelque chose est faux, ou parce qu'il n'a pas d'intérêt, ou pour n'importe quelle autre raison. Si rien n'était ajouté ou enlever un programme lors d'un **BREAK**, alors il faudrait repartir en frappant :

```
CONTINUE
```

Si quelque chose a changé et si le programme ne peut pas continuer, alors l'ordinateur répond :

```
Bad Line
```

INTERRUPTION MOMENTANEE

CTRL et **F5** permet d'interrompre momentanément le déroulement du Basic. La pression de n'importe quelle touche relance celui-ci. Ceci est très utile pour arrêter momentanément le défilement d'un listing ou l'affichage d'un répertoire.

RESET

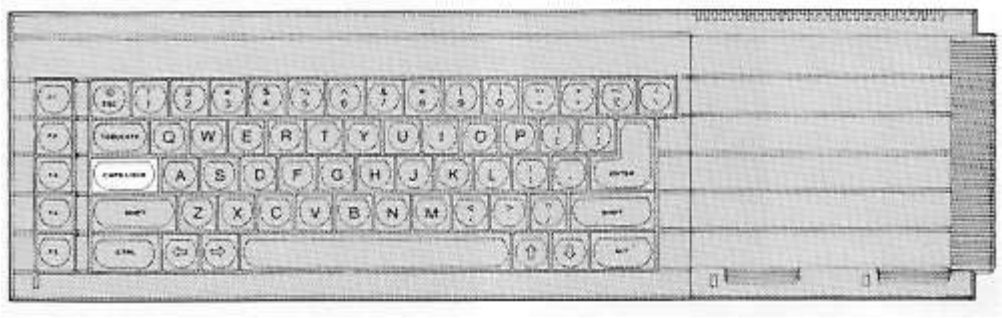
Ceci n'est pas une touche, mais un petit bouton-poussoir situé sur le côté droit du QL. Il est placé volontairement à cet endroit, endure des touches courantes, parce que ces effets sont plus dramatiques que le **BREAK**. Si vous n'obtenez pas satisfaction avec un **BREAK**, alors appuyez sur le bouton **RESET**. Vous aurez presque la même chose qu'en coupant le contact de l'ordinateur, et en le retranchant. Vous obtenez un véritable redémarrage.

SHIFT

Il y a deux touches **SHIFT** parce qu'elles sont utilisées fréquemment, et il est nécessaire qu'elle soit accessible à n'importe quelle main. Appuyez sur **SHIFT** et frappez quelques caractères alphabétiques. Vous obtiendrez des caractères MAJUSCULES

Appuyez sur la touche **SHIFT**, et frappez quelques autres touches, non alphabétiques, vous obtiendrez le symbole inscrit en haut de la touche. Sans la touche **SHIFT**, vous obtenez des lettres minuscules ou le symbole inscrit au bas de la touche.

ALPHA

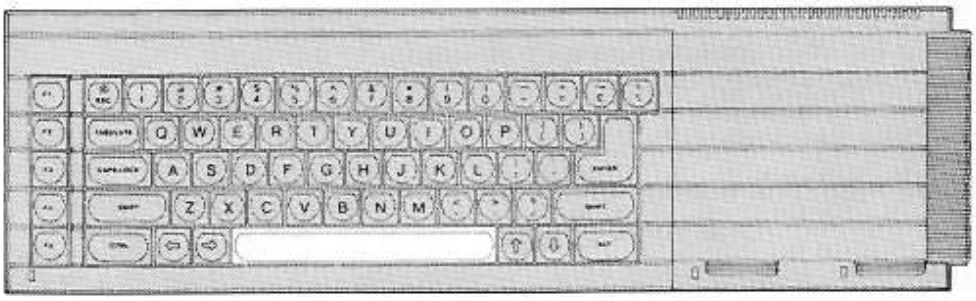


Cette touche fonctionne comme un commutateur. Frappez là une fois pour toutes, et les touches affichant des lettres seront bloquées dans un mode unique : majuscules ou minuscules.

- Frappez quelques touches alphabétiques
- Enfoncez la touche **ALPHA**
- Frappez quelques touches alphabétiques

Vous constaterez que le mode change et reste jusqu'à ce que vous enfonciez à nouveau la touche **ALPHA**.

BARRE D'ESPACEMENT

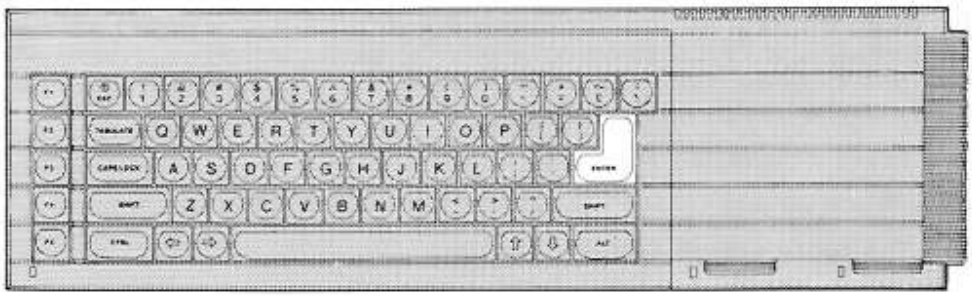


La longue touche située au bas du clavier génère des espaces. C'est une touche très importante en SuperBASIC, comme vous le verrez au chapitre deux.

EFFAÇAGE

La touche située à la gauche de la barre d'espace utilisée en même temps que la touche **CTRL** agit comme une gomme. Vous devez appuyer sur la touche **CTRL** en même temps que sur le curseur. À chaque frappe le caractère précédent sera supprimé.

ENTREE



Le système a besoin de savoir quand vous avez terminé l'écriture d'un message complet ou d'une instruction. Quand vous avez frappé quelque chose de complet tel que **RUN**, vous frappez la touche ENTREE pour valider la commande.

Cette touche étant souvent utilisée, nous lui avons attribué un symbole spécial : ⏎

Nous utiliserons ce symbole par commodité, pour une meilleure présentation, et par économie de place. Testez la touche ⏎ (ENTREE) en tapant :

```
PRINT " Correct " ⏎
```

Si vous n'avez pas fait de faute, le système vous répondra :

```
Correct
```

AUTRES TOUCHES IMMEDIATEMENT UTILISABLES

*	Multiplier	+	Additionner
_	Souligné	=	Devient égal à (utilisé avec l'instruction LET)
“	Guillemets	‘	apostrophe
,	Virgule	!	exclamation
;	Point-virgule	&	apostrophe
:	Deux points	.	Point (décimal ou final)
\	Slash inversé	$	dollar
(	Parenthèse ouvrante	)	Parenthèse fermante

MAJUSCULES ET MINUSCULES

SuperBASIC reconnaît les commandes, qu'elles soient écrites en caractères majuscules ou minuscules. Par exemple, la commande SuperBASIC pour effacer l'écran est **CLS** et peut-être tapée :

- CLS ⏎
- Cls ⏎
- clS ⏎

Ces trois expressions sont correctes et ont le même effet.

EMPLOI DES GUILLEMETS

Les guillemets servent à définir un mot, ou une phrase, ou une chaîne de caractères. Essayez :

```
PRINT " ce travail "
```

L’ordinateur répondra :

```
ce travail
```

Les guillemets ne sont pas affichés, mais ils indiquent qu'un texte doit être imprimé et ils le délimitent exactement. Si vous souhaitez utiliser les guillemets eux-mêmes dans une chaîne de caractères, alors vous pouvez remplacer les guillemets du début et de fin par des apostrophes. Exemple :

```
PRINT ‘le symbole des guillemets est " ‘
```

L’ordinateur répond :

```
le symbole des guillemets est "
```

ERREURS COURANTES DE FRAPPE

La touche zéro est avec les autres touches numériques en haut du clavier, et elle est légèrement plus étroite. La lettre « O » est parmi les autres lettres. Veillez à utiliser le symbole approprié.

On confond aussi facilement la touche «Un » (1) parmi les touches numériques et la lettre « I » parmi les lettres. Veillez à l'éviter.

GARDER « SHIFT » APPUYE

Quand vous utilisez la touche **SHIFT**, appuyez sur cette touche pendant que vous en enfoncez une autre, de sorte que la touche **SHIFT** soit en contact avant l'autre.

La même règle s'applique aux touches **CTRL** et **ALT**, qui sont utilisées en conjonction avec d'autres, mais vous n'en avez pas besoin maintenant.

CHARGEMENT DE PROGRAMMES

Tapez ces deux simples instructions :

```
CLS ↵
PRINT 'Hello' ↵
```

A strictement parler, ceci constitue un programme informatique, cependant, c'est le programme chargé qui est important programmation. Les instructions ci-dessus sont exécutées instantanément dès que vous appuyez sur la touche ENTREE.

Maintenant, rentrez le programme avec les numéros de lignes :

```
10   CLS ↵
20   PRINT 'HELLO' ↵
```

Cette fois-ci, rien ne se passe, si ce n'est que le programme s'affiche dans la partie supérieure de l'écran. Cela signifie qu'il est accepté, sa grammaire et sa syntaxe étant correctes. Il est conforme aux règles du SuperBASIC, mais il n'a pas encore été exécuté, il est simplement chargé. Pour l'exécuter, tapez :

```
RUN ↵
```

La différence entre les commandes directes pour une exécution immédiate et une séquence de chargement des instructions est expliquée dans le prochain chapitre. Pour l'instant, vous pouvez expérimenter les idées précédentes et encore deux autres :

```
LIST ↵
```

provoque un affichage du programme à l'écran.

```
NEW ↵
```

provoque une suppression du programme, de sorte que vous pouvez ensuite travailler dans un autre.

TESTS SUR LE CHAPITRE UN

A faire soi-même

Vous pouvez totaliser un maximum de 16 points dans le test suivant. Calculez votre score avec les réponses sur la page suivante.

1. *Dans quelles circonstances pouvez-vous utiliser le BREAK ?*
2. *Où est le bouton RESET ?*
3. *Quel est l'effet du bouton RESET ?*
4. *Nommer de différence entre une touche SHIFT et la touche ALPHA.*
5. *Comment pouvez-vous supprimer un caractère faux que vous venez juste de taper ?*
6. *Quel est le symbole utilisé pour la touche ENTREE ?*
7. *Quel est l'effet sur la commande dans les questions 8 à 11 ?*
8. `CLS ↵`
9. `RUN ↵`
10. `LIST ↵`
11. `NEW ↵`

12. Est-ce que les touchants ont le même effet si vous les tapez en majuscules ?
13. Que veut dire la partie des mots-clés que le QL indique en majuscules ?

REPONSE AU TEST SUR LE CHAPITRE 1.

1. Utilisez **BREAK** pour abandonner un programme en cours parce que :
   - quelque chose est faux, et vous ne comprenez pas pourquoi,
   - il est sans intérêt,
   - tout autre problème. (3 pts)

2. Le bouton **RESET** est sur la droite de l'ordinateur.

3. L'effet du bouton **RESET** est le même que celui de l'arrêt suivi d'une mise en marche immédiate de l'ordinateur.

4. La touche SHIFT :
   - est effective seulement pendant que vous appuyez dessus alors que la touche **ALPHA** reste effective après que vous ayez appuyé dessus. (1 pt)
   - la touche **SHIFT** concerne tous les caractères alphabétiques, numériques et symboliques, mais la touche **ALPHA** ne concerne que les lettres. (1 pt)

5. La touche **CTRL** ← supprime le caractère prévu juste à gauche du curseur.

6. La touche ↵ (ENTREE) provoque l'entrée d'un message ou d'une instruction dans l'ordinateur.

7. Nous utilisons ↵ pour la touche ENTREE.

8. `CLS` ↵ provoque le nettoyage d'une partie de l'écran.

9. `RUN` ↵ provoque l'exécution d'un programme chargé en mémoire.

10. `LIST` ↵ provoque l'affichage à l'écran d'un programme chargé en mémoire.

11. `NEW` ↵ nettoie la mémoire et la rend apte à recevoir un nouveau programme.

12. Les caractères de SuperBASIC sont reconnus en majuscules ou en minuscules.

13. La partie d'un mot-clé imprimé en majuscules est l'abréviation autorisée.

Vérifiez votre score :

14-16 : Très Bien - passez à la suite.

12-13 : Bien, mais relisez quelques parties du chapitre 1.

10-11 : Acceptable, relisez des parties du chapitre 1 et refaites le test.

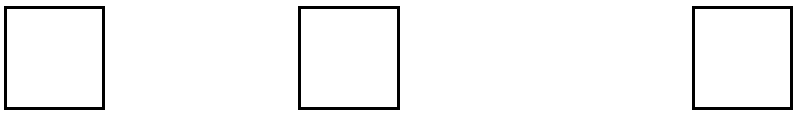
0-9 : Revoyez soigneusement tout le chapitre, et refaites le test.

CHAPITRE 2

PROGRAMMATION

NOMBRES, NOMS ET CASES

Les ordinateurs ont besoin de charger des données telles que des nombres. On peut imaginer que le chargement s'effectue dans des cases :



Bien que vous ne puissiez pas les voir, il faut que vous donniez un nom à chaque case. Supposons que vous désirez effectuer le simple calcul suivant :

Un éleveur de chiens à neuf chiens à nourrir pendant 28 jours. Chacun reçoit une boîte de « Médor » par jour. Calculez le nombre de boîtes.

Une façon de résoudre ce problème nécessite trois cases pour :

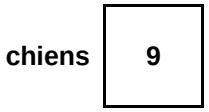
- nombre de *chiens*
- nombre de *jours*
- nombre de *boîtes*

SuperBASIC vous autorise à choisir les noms de chaque case et vous pouvez choisir :

Vous pouvez créer une case, la nommer et charger un nombre de temps avec la simple instruction :

```
LET chiens = 9 ↵
```

Ceci va créer une case mémoire, nommé chiens, et placer dedans le nombre 9 :



Le mot **LET** a une signification spéciale en SuperBASIC, il est appelé mot-clé. Le SuperBASIC a plusieurs autres mots-clés que vous étudierez plus tard. Vous devez veiller à ce que l'espace après **LET** et les autres mots-clés soient présents, car SuperBASIC vous autorise à choisir le nom de chaque case avec une certaine liberté ; **LET** chiens serait un nom de casse correct.

Le mot-clé **LET** est optionnel en SuperBASIC, et de ce fait :

```
LETchiens = 9
```

est valide, mais se réfère à une case nommée LETchiens et non chiens.

De la même façon, en français, les noms, les nombres et les mots-clés sont séparés entrent eux par des espaces, s'ils ne le sont pas par des caractères spéciaux. Vous pouvez vérifier qu'une case mémoire existe en tapant :

```
PRINT chiens ↵
```

L'écran affichera le contenu de la case : **9**

À nouveau, attention à l'espace après **PRINT**.

Pour résoudre le problème nous pouvons écrire un programme qui est une succession d'instructions : vous pouvez maintenant comprendre les deux premières :

```
LET chiens = 9 ↵
LET jours = 28 ↵
```

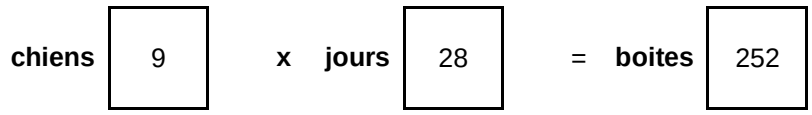
Ceci provoque la création de deux cases nommées chiens et jours, et leur donne les valeurs 9 et 28.

L'instruction suivante doit réaliser une multiplication, dans le symbole de l'ordinateur est *, qui place le résultat dans une nouvelle case appelée boîtes :

```
LET boites = chiens * jours ↵
```

1. L'ordinateur recherchait les valeurs 9 et 28 des cases nommées chiens et jours.
2. Le nombre 9 est multiplié par 28.
3. Une nouvelle qui avait créé et nommée boites.
4. Le résultat de la multiplication est placé dans la case nommée boîtes.

Tout ceci peut paraître très détaillé, mais vous avez besoin de comprendre ces notions qui sont très importantes. Le résultat peut être représenté ainsi :



Il ne reste plus qu'à demander à l'ordinateur d'afficher le résultat en tapant :

```
PRINT boites ↵
```

qui donnera à l'écran :

```
252
```

En résumé, le programme est :

```
LET chiens = 9 ↵
LET jours = 28 ↵
LET boites = chiens * jours ↵
PRINT boites ↵
```

Naturellement, vous obtiendriez ce résultat plus facilement avec une calculatrice ou un crayon et un papier. Vous pourriez obtenir rapidement avec le QL en tapant :

```
PRINT 9 * 28
```

qui donnera le résultat à l'écran. Cependant, les notions que nous avons exposées sont les points essentiels de démarrage d'une programmation en SuperBASIC. Elles sont aussi essentielles qu'elles apparaissent dans beaucoup de langages informatiques et ont reçu des noms spéciaux.

1. Les noms tels que chiens, jours et boîtes sont appelés les identifiants.
2. Une simple instruction telle que :
   ```
   LET chiens = 9 ↵
   ```
   est appelée une affectation.
3. Le nombre associé au nom de la case est appelé la variable. L'exécution de l'affectation ci-dessus charge la valeur 9 dans la case identifiée par l'identifiant chiens. Une affectation telle que :
   ```
   LET chiens = 9 ↵
   ```
   est une instruction de traitement interne à la machine, mais le texte affiché est statique et il utilise le signe = emprunté aux mathématiques. Il serait plus juste de penser ou de dire (mais non de taper) :
   ```
   LET chiens devient 9 ↵
   ```
   et d'écrire :
   ```
   chiens ← 9
   ```

Le rôle du signe = dans une affectation LET n'est pas le même qu'en mathématiques. Par exemple, si un autre chien arrive, vous pouvez écrire :

```
LET chiens = chiens + 1 ↵
```

Mathématiquement, ceci n'est pas très correcte, mais en informatique, c'est une opération simple. Si la valeur de chiens étaie 9 avant l'opération, après l'opération, elle sera 10.

Tester ceci en tapant :

```
LET chiens = 9 ↵
PRINT chiens ↵
LET chiens = chiens + 1 ↵
PRINT chiens ↵
```

L'affichage sera successivement le suivant :

```
9
10
```

ce qui montre que la valeur finale de la case est :

chiens	**10**

Une bonne manière de comprendre ce qui se passe dans les cases et de faire ce que l'on appelle un « déroulement manuel ». Vous examinez simplement chaque instruction à son tour et écrivez les valeurs qui en résultent pour montrer comment les cases sont créées, les valeurs qu'elles contiennent, et comment évoluent les valeurs pendant l'exécution du programme :

	chiens	jours	boites
`LET chiens = 9 ↵`	**9**		
`LET jours = 28 ↵`	**9**	**28**	
`LET boites = chiens * jours ↵`	**9**	**28**	**252**
`PRINT boites ↵`	**9**	**28**	**252**

L'écran affiche 252.

Vous pouvez remarquer que, depuis le début, un nom de variable a toujours été utilisé en premier à gauche de l'affectation **LET**. Une fois la case mémoire créée et contenant une valeur, le nom de la variable correspondante peut être utilisé dans la partie droite d'une affectation.

Maintenant, supposons que vous désirez encourager un enfant à gagner de l'argent. Vous lui donnez de barres de chocolat pour chaque franc gagné. Supposons que vous essayiez de calculer ainsi :

```
LET barres = francs * 2 ↵
PRINT barres ↵
```

Tel quel programme vous ne pouvez avoir de résultats, puisque vous ne savez pas combien de francs ont été gagnés.

barres	francs

LET barres = francs * 2 ↵	**?**	**?**

Nous avons fait ici une erreur volontaire en utilisant francs à droite d'une affectation **LET** sans avoir créé et charger cette case. Votre QL va chercher dans sa mémoire la variable francs. Il ne la trouvera pas et conclura qu'il s'agit d'une erreur de programme et enverra un message d'erreur. Nous disons que la variable francs n'a pas été initialisée. Le programme sera correct et vous commencez ainsi :

	barres	francs
LET francs = 7 ↵	**7**	
LET barres = francs * 2 ↵	**7**	**14**

Le programme se déroule correctement et donne en sortie : **14**

PROGRAMME CHARGE

Taper des instructions sans numéro de ligne peut produire le résultat désiré, mais cette méthode n'est pas satisfaisante pour deux raisons (sauf pour l'introduction) :

1. Le programme ne peut s'exécuter qu'à la vitesse où vous pouvez taper. Ce n'est pas très impressionnant pour une machine capable de faire des millions d'opérations à la seconde.
2. Les instructions individuelles ne sont pas chargées après exécution, aussi ne pouvez-vous pas exécuter le programme à nouveau, ou corriger une erreur sans retaper la même chose en entier.

Charles Babbage, un pionnier de l'informatique, vivant au XIXe siècle, avait pressenti qu'un ordinateur de qualité devait pouvoir charger les instructions aussi bien que les données dans des cases mémoire. Ces instructions seraient alors exécutées rapidement, en séquence, sans autre intervention humaine.

Les instructions du programme seront chargées mais non exécutées si vous utilisez des numéros de ligne.

Essayez ceci :

```
10 LET prix = 15 ↵
20 LET nb = 7 ↵
30 LET coût = prix * nb ↵
40 PRINT coût ↵
```

Il ne se passe rien de visible, mais la totalité du programme et charger à l'intérieur de la machine. Vous pouvez l'exécuter en tapant :

```
RUN ↵
```

Et la sortie : 105 apparaîtra.

L'avantage de cette méthode est de pouvoir éditer ou rajouter des lignes à un programme, en tapant un minimum de ligne.

ÉDITION D'UN PROGRAMME

Plus tard, vous apprendrez toutes les caractéristiques d'édition du SuperBASIC, mais même à ce stade, vous pouvez aisément faire trois choses :

- remplacer une ligne,
- insérer une nouvelle ligne,
- supprimer une ligne.

REMPLACER UNE LIGNE

Supposons que vous souhaitiez modifier le programme en cours parce que le prix a changé, et est passé à 20. Il suffit de retaper la ligne 10 :

```
10 LET prix = 20 ↵
```

Cette ligne va remplacer la ligne 10 précédente. Assurons-nous que les autres lignes sont encore chargées, et testant le programme en tapant :

```
RUN ↵
```

et la nouvelle réponse `140` apparaît.

INSERER UNE LIGNE

Supposons que vous souhaitiez insérer une ligne juste avant la dernière, pour écrire les mots « prix total ». Cette situation n'arrive souvent, et c'est pourquoi nous choisissons généralement les nombres, 10, 20,30… Pour permettre désinsertion entre les lignes déjà écrites.

Pour écrire entre les lignes 30 et 40, tapez :

```
35 PRINT " prix total " ↵
```

Cette ligne s'insérera juste avant la ligne 40. Le système permet un grand choix de numéro de ligne, dans l'échelle de 1 à 32 768. Il est difficile de prévoir à l'avance les changements nécessaires.

Maintenant tapez :

```
RUN ↵
```

et la nouvelle sortie sera :

```
prix total
140
```

SUPPRIMER UNE LIGNE

Vous pouvez supprimer la ligne 35 en tapant :

```
35 ↵
```

Cela a le même effet que l'écriture d'une ligne en blanc qui remplace la précédente.

SORTIE – PRINT

Il faut noter la commodité de l'instruction **PRINT**. Vous pouvez imprimer un texte en utilisant des guillemets ou des apostrophes.

```
PRINT " barres de chocolat " ↵
```

Vous pouvez imaginer les valeurs de variable continue dans les cases mémoire :

```
PRINT barres ↵
```

sans utiliser des guillemets.

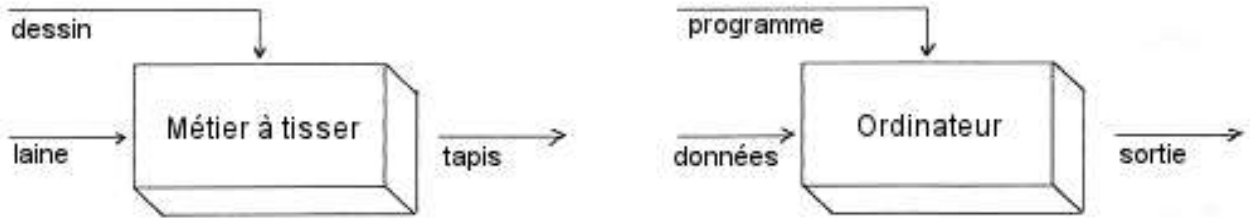
Vous verrez par la suite les différentes variations des applications de l'instruction **PRINT**. Vous pourrez disposer le texte sur écran, à l'endroit que vous aurez choisi.

Mais pour le moment, les deux possibilités suivantes sont suffisantes :

- écriture d'un texte,
- inscription des valeurs de variables (contenu de cases mémoire)

INPUT – READ – DATA

Un fabricant de tapis a besoin de laine en entrée. Il fabrique des tapis en sortie :



En changeant de type de laine, on obtient des tapis différents.

Il existe le même type de relations dans un ordinateur. Cependant, s'il a donné est entré dans une case mémoire à l'aide d'une instruction **LET**, il a deux inconvénients, lorsqu'il s'agit de programmes un peu élaborés :

- écrire à l'instruction LET est laborieux,
- changer les valeurs en entrée est aussi laborieux.

Il est possible d'entrer les données en cours d'exécution du programme. L'instruction **INPUT** provoque un arrêt du programme qui se met en attente de la frappe d'un mot au clavier.

Taper d'abord :

```
NEW ↵
```

Ainsi, le programme en cours sera effacé, et le QL est prêt pour un nouveau programme. Maintenant tapez :

```
10 LET prix = 15 ↵
20 PRINT "Ques est le nombre de stylos ?" ↵
30 INPUT nb ↵
40 LET cout = prix * nb ↵
50 PRINT cout ↵
RUN ↵
```

Le programme s'arrête à la ligne 30 et vous pouvez taper le nombre de stylos que vous désirez. Tapez le nombre :

```
4 ↵
```

N'oubliez pas la touche ENTREE. La sortie sera :

```
60
```

L'instruction **INPUT** nécessite un nom de variable de manière que le système sache où mettre la donnée que vous avez entrée au clavier. Lorsque vous entrez 4, l'effet de la ligne 30 est le même qu'une instruction **LET**. **INPUT** est plus pratique pour qu'il y ait dialogue entre l'ordinateur et l'utilisateur. Toutefois, les instructions **LET** et **INPUT** ne sont utilisées que pour de petites quantités de données. Pour de plus grandes quantités, on utilise autre chose. SuperBASIC comme la plupart des BASIC mettent à votre disposition une autre méthode d'entrée connue sous le nom de **READ** (lire) à partir d'une instruction **DATA**. Nous pouvons rentrer le programme ci-dessous dans une forme nouvelle, qui donnera le même résultat, mais sans aucun arrêt. Essayez ceci :

```
NEW ↵
10 READ prix, nb ↵
20 LET cout = prix * nb ↵
30 PRINT cout ↵
```

```
40 DATA 15, 4 ↵
RUN ↵
```

Vous avez en sortie :

```
60
```

comme auparavant.

Quand la ligne 10 est exécutée, le système cherche dans le programme une instruction **DATA**. Il utilise alors les valeurs de l'instruction **DATA** pour les variables de l'instruction **READ**, exactement dans le même ordre. On place généralement les instructions **DATA** à la fin du programme. Elles sont utilisées par le programme, mais elles ne sont pas exécutées de la même façon que les autres. Les instructions **DATA** peuvent être, n'importe où dans le programme, mais il est préférable de les mettre à la fin. Elles sont nécessaires, mais ne représente pas la partie importante du programme. Les règles concernant **READ** et **DATA** sont les suivantes :

1. Toutes les instructions **DATA** sont considérées comme une seule longue suite d'articles. Ces articles peuvent être des nombres ou des mots.
2. Chaque fois qu'une instruction **READ** est exécutée, les articles nécessaires sont copiés de l'instruction **DATA** dans les variables nommées dans l'instruction **READ**.
3. Le système conserve une trace des articles lus à l'aide d'un enregistrement interne. Il un programme demande la lecture de plus d'articles qu'il en existe dans l'instruction **DATA**, une erreur sera signalée.

IDENTIFIANTS

Vous avez utilisé les noms pour les cases mémoire tels que chiens, barres. Vous devez choisir ces mots en vous conformant à certaines règles :

- Un nom ne doit pas inclure d'espace.
- Un nom doit commencer par une lettre.
- Un nom doit être composé uniquement de lettres, nombres, ainsi que des signes $ % et _.
- Les symboles $ et % ont des fonctions spéciales, qui seront expliqués plus loin, mais vous pouvez utiliser le souligné _ pour faire des noms tels que : aliment_chien, salaire_mensuel_total.

SuperBASIC ne distingue pas les caractères majuscules et minuscules, aussi les noms tels que BOITES et boîtes sont les mêmes. Le nombre maximum de caractères d'un nom est 255.

Les noms construits suivant ses règles sont appelés des identifiants. Les identifiants sont utilisés pour d'autres fonctions en SuperBASIC et vous devez les connaître. Les règles permettent une grande liberté dans le choix des noms, aussi pouvez-vous écrire votre programme de sorte qu'il soit plus facile à comprendre. Des noms tels que total, compte, stylos sont plus parlants que Z, P, Q.

TEST SUR LE CHAPITRE 2

Vous pouvez gagner un maximum de 21 points pour ce test. Vérifiez votre score avec les réponses de la page suivante.

1. Comment vous représentez-vous un nombre chargé en mémoire ?
2. Indiquez deux façons de charger une valeur dans une case mémoire (2pts).
3. Comment pouvez-vous lire la valeur d'une case mémoire ?
4. Quel est le nom usuel du contenu d'une case mémoire
5. À quel moment une case mémoire prend-elle sa première valeur ?
6. Une variable est ainsi nommée parce que ces valeurs peuvent varier pendant l'exécution du programme. Quel est la cause habituelle de telles variations ?
7. Le signe = dans une instruction **LET** ne signifie pas «égal » comme en mathématiques. Que signifie-t-il ?
8. Que se passe-t-il quand vous entrez une instruction non numérotée ?
9. Quelle est la fonction des guillemets dans une instruction **PRINT** ?
10. Que se passe-t-il quand vous n'utilisez pas les guillemets dans une instruction **PRINT** ?
11. Quelles différences entre les instructions **INPUT** et **LET** ?
12. Quel type d'instruction du programme n'est jamais exécuté ?
13. Quelle est la fonction de l'instruction **DATA** ?
14. Comment appelle-t-on une case mémoire (ou variable) ?
15. Donner trois exemples d'identifiants utilisant des lettres, des lettres et des nombres, des lettres et des caractères (3 pts).
16. Pourquoi l'espace est-il particulièrement important en SuperBASIC ?
17. Pourquoi est-il important de choisir librement les identifiants d'un programme ?

REPONSES AU TEST DU CHAPITRE 2

1. Un nombre chargé en mémoire comme une case mémoire que vous pouvez nommer et remplir d'un nombre.
2. Une instruction **LET** qui utilise un certain nombre pour la première fois provoque la création d'une case mémoire ayant ce nom, par exemple : **LET** compte = 1 (1pt)

   Une instruction **READ** qui utilise un nom pour la première fois aura le même effet, par exemple : **READ** compte (1 pt)
3. Vous pouvez connaître la valeur d'une case mémoire avec une instruction **PRINT**.
4. Le nom théorique d'une case mémoire est « variable » parce que ses valeurs peuvent évoluer au cours du déroulement du programme.
5. Une variable reçoit sa première valeur quand elle est utilisée pour la première fois dans une instruction **LET**, **INPUT** ou **READ**.
6. C'est l'exécution d'une instruction **LET** qui change généralement la valeur de la variable.
7. Le signe = dans une instruction **LET** représente l'opération « évaluer l'expression qui est située à droite et placer le résultat dans la case mémoire nommée à gauche (du signe =) ». Cela signifie : « le côté gauche doit devenir égal au côté droit ».
8. Une instruction non numérotée s'exécute immédiatement.
9. Une instruction numérotée ne s'exécute pas immédiatement. Elle est chargée en mémoire.
10. Dans une instruction **PRINT**, les guillemets entourent le texte qui doit être affiché.
11. Si les guillemets ne sont pas utilisés, c'est la valeur de la variable qui est affichée.
12. Une instruction **INPUT** interrompt le programme de façon que vous puissiez entrer les données au clavier.
13. Les instructions **DATA** ne sont jamais exécutées.
14. Elles sont utilisées pour procurer des valeurs aux variables des instructions **READ**.
15. Le terme technique pour le nom d'une case mémoire est « identifiant ».
16. Exemples de réponses :
    - jour
    - jour_23
    - jour_de_la_semaine
17. La barre d'espacement est particulièrement importante, car elle génère des espaces avant ou après les mots-clés afin d'éviter qu'ils soient considérés comme des identifiants choisis par l'utilisateur.
18. Il est important de choisir librement vos identifiants, car ils vous aident à écrire des programmes plus faciles à comprendre. De tels programmes risquent moins les erreurs et sont plus faciles à mettre au point.

Vérifiez votre score :

18-21 : Très bien. Continuez la lecture.

16-17 : Bien, mais relisez quelques parties du chapitre 2.

14-15 : Convenable, mais relisez quelques parties du chapitre 2 et refaites le test.

0-13 : Vous devez revoir attentivement le chapitre 2 et refaire le test.

PROBLEMES SUR LE CHAPITRE 2

1. Exécutez un déroulement manuel pour montrer les valeurs de toutes les variables dans le programme suivant.

2. Écrivez et testez un programme qui calcule l'aire d'un tapis de 3 m de large de 4 m de long. Utilisez comme nom de variable : longueur, largeur, aire.

3. Réécrivez ce programme en utilisant deux instructions **INPUT** à la place des instructions **LET**.

4. Réécrivez ce programme de sorte que les données en entrée (3 et 4) apparaissent par une instruction **DATA** au lieu d'une instruction **LET**.

5. Réécrivez la solution du problème 2 en utilisant une autre méthode pour les données en entrée. Utilisez **READ** et **DATA** si vous aviez d'abord utilisé **LET** et vice versa.

6. Bill et Ben décide de faire un jeu. Chacun sort de son portefeuille tous ses billets, et ils se les donnent l'un dans l'autre. Écrivez un programme qui simule entièrement cette situation avec des instructions **LET** et **PRINT**.
   Utilisez une troisième personne, Sue, qui prend l'argent de Bill, pendant que Bill prend celui de Ben.

7. Réécrivez le problème numéro 6 en exprimant les deux nombres qui doivent être échangés à l'aide d'instructions **DATA**.

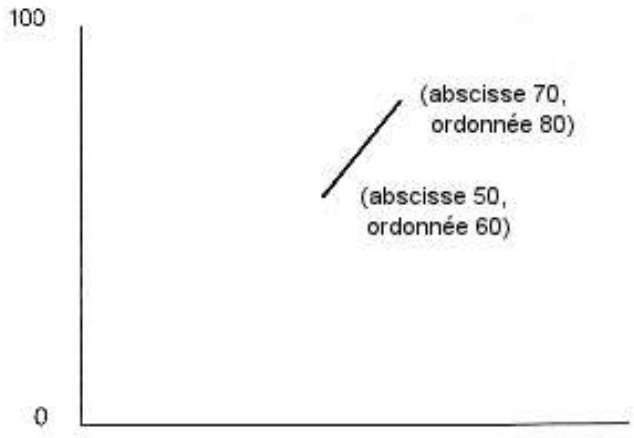
CHAPITRE 3

LES TRACES A L'ECRAN

L'ECRAN

Dans le but de permettre de type d'unité d'affichage (téléviseur et moniteur) de modes d'écran sont disponibles. Cependant, il ne serait pas bon qu'un programme écrit pour tracer des cercles et des carrés dans un mode produisent des ellipses et des rectangles dans l'autre mode (comme le font certains systèmes). Aussi avons-nous mis à votre disposition un système d'échelles graphiques qui évitent ces problèmes. Vous choisissez simplement une échelle verticale, et vous travaillez avec. L'autre type de graphique, orienté pixel, est aussi disponible et il est décrit dans un prochain chapitre.

Supposons, par exemple, que nous choisissions une échelle verticale de 100 et que nous souhaitions tracer une ligne du point (50,60) au point (70,80).



UNE LIGNE COLOREE

Nous devons préciser quatre paramètres :

- L'ECHELLE (verticale)
- PAPER (couleur du fond)
- INK (couleur des traits)
- LINE (point de départ et point d'arrivée)

Le programme suivant trace une droite comme sur la figure ci-dessus, rouge (couleur code 2), sur un fond blanc (couleur code 7).

```
NEW ↵
10 PAPER 7 : CLS ↵
20 INK 2 ↵
30 LINE 50,60 TO 70,80 ↵
RUN ↵
```

Dans la ligne 20, il faut noter que la couleur du papier est d'abord sélectionnée, mais elle ne devient effective qu'après une prochaine commande, comme **CLS**, afin de nettoyer l'écran avant d'avoir la couleur demandée.

MODES ET COULEURS

Jusqu'ici, peu importe le mode écran que vous utilisez, cependant les couleurs sont affectées par le choix d'un mode :

- MODE 8 permet huit couleurs de base.
- MODE 4 permet quatre couleurs de base.

Les couleurs ont des codes indiqués ci-dessous :

Code	Résultat	
	8 couleurs	4 couleurs
0	Noir	Noir
1	Bleu	Noir
2	Rouge	Rouge
3	Magenta	Rouge
4	Vert	Vert
5	Cyan	Vert
6	Jaune	Blanc
7	Blanc	Blanc

Par exemple, **INK** 3 donnerait magenta en mode 8 et rouge en mode 4.

Nous expliquerons dans un prochain chapitre comment les couleurs de base peuvent être mélangées de différentes manières pour produire des tons et dégradés étonnants.

NOMBRES ALEATOIRES

Vous pouvez obtenir des résultats intéressants avec les nombres aléatoires qui peuvent être générés avec la fonction RND. Par exemple :

```
PRINT RND (1 TO 6)↵
```

affichera un nombre compris entre 1 et 6, comme si vous aviez jeté un dé assis face. Le programme suivant illustre ceci :

```
NEW ↵
10 LET die = RND(1 TO 6) ↵
20 PRINT die ↵
RUN ↵
```

Si vous exécutez le programme plusieurs fois, vous obtenez des nombres différents.

Vous pouvez obtenir des nombres aléatoires dans n'importe quel intervalle. Par exemple :

```
RND(0 TO 100)
```

produira un nombre qui peut être utilisé en échelle graphique.

Vous pouvez réécrire la droite du programme en créant une droite aléatoire dans une couleur aléatoire. Lorsque l'intervalle des nombres aléatoires commence à zéro, vous pouvez omettre le premier nombre et écrire :

```
NEW ↵
10 PAPER 7 : CLS ↵
20 INK RND(5) ↵
30 LINE 50,60 TO RND(100),RND(100) ↵
RUN ↵
```

Ceci produit une droite qui part d'un point situé près du centre de l'écran et se termine en un point aléatoire. L'intervalle des couleurs possibles dépend du mode sélectionné. Vous remarquerez qu'un intervalle de nombres « I to I » revient souvent en SuperBASIC.

BORDURES

La partie de l'écran dans laquelle vous dessinez des droites et créez d'autres figures est appelé une fenêtre. Plus tard, vous verrez comment changer la taille et la position d'une fenêtre ou bien créer d'autres fenêtres. Pour l'instant, nous nous contenterons de dessiner une bordure autour de la fenêtre existante. La plus petite surface colorée que vous puissiez obtenir à l'écran êtes appelés un **pixel**. En mode 8, appelé **mode basse résolution**, il y a 256 positions possibles pour les pixels dans la largeur de l'écran, et 256 dans la hauteur. En mode 4, **appelé mode haute résolution**, il y a 512 pixels en largeur, et 256 en hauteur. Ainsi, la taille d'un pixel dépend du mode.

Vous pouvez construire une bordure autour de l'arête d'une fenêtre en tapant, par exemple :

```
BORDER 4,2 ↵
```

Ceci créera une bordure de 4 pixels de large de couleur rouge (code 2). La bordure diminue la taille effective de la fenêtre. Par conséquent, aucun texte, ni graphique ne pourra s'intercaler entre les deux bordures. Le seul moyen d'y parvenir serait de créer une autre bordure qui écraserait la précédente.

UNE SIMPLE BOUCLE

Les ordinateurs sont capables de faire des choses à une très grande vitesse, mais il ne serait pas possible d'exploiter cette étonnante puissance si chaque action nécessitait l'écriture d'une instruction. Un chef de travaux, dans le bâtiment, a le même problème. Lorsqu'il veut qu'un ouvrier pose une centaine de parpaings, il ne lui donne pas 100 ordres séparés.

La manière habituelle d'achever une boucle de répétition en BASIC est d'utiliser l'instruction **GO TO** (ou **GOTO**, c'est la même chose), comme suit :

```
NEW ↵
10 PAPER 6 : CLS ↵
20 BORDER 1,2 ↵
30 INK RND(5) ↵
40 LINE 50,60 TO RND(100),RND(100) ↵
50 GOTO 30 ↵
RUN ↵
```

Vous préférez peut-être ne pas rentrer ce programme car SuperBASIC propose un meilleur moyen de faire des répétitions. Notez le niveau de répétition de chaque ligne :

10 20	Partie fixe - pas de répétition
30 40	Partie variable - répétition
50	Contrôle

Vous pouvez réécrire le programme ci-dessus en éliminant l'instruction GOTO et en la remplaçant par REPEAT et END REPEAT autour de la partie à répétition.

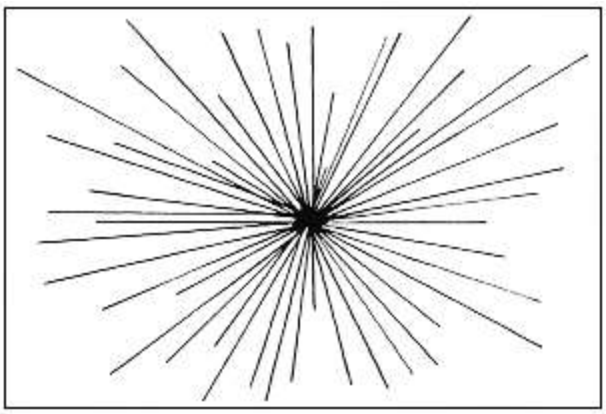
```
NEW ↵
10 PAPER 6 : CLS ↵
20 BORDER 1,2 ↵
30 REPEAT star ↵
40   INK RND(5) ↵
50   LINE 50,60 TO RND(100),RND(100) ↵
60 END REPEAT star ↵
RUN ↵
```

Nous avons donné à la structure REPEAT un nom : star. La structure se compose de deux lignes :

```
REPeat star
END REPeat star
```

et ce qui est entre les deux s'appelle le contenu de la structure.

Ce programme produirait des droites de couleurs différentes, indéfiniment, pour faire une étoile comme ci-dessous :



Le programme STAR

Vous pouvez l'interrompre en faisant un break. Appuyez tout d'abord sur la touche CTRL, puis en maintenant celle-ci enfoncée appuyez sur la touche SPACE.

SuperBASIC vous fournit la méthode pour stopper les répétitions. Imaginez que vous êtes dans un programme qui boucle, comment en sortir ? La réponse est d'utiliser une instruction **EXIT**. Mais il faut un motif pour en sortir. Vous pourriez rajouter une ligne de modification de programme (ne tapez pas **NEW**) :

```
40 INK RND (0 TO 6) ↵
```

Ainsi si RND produit 6, la couleur de l'encre devient la même que celle du fonds et vous ne voyez plus rien. Cela pourrait être la raison de terminer la répétition. Nous modifierons donc le programme de la façon suivante :

```
NEW ↵
10 PAPER 6 : CLS ↵
20 BORDER 1 ,2 ↵
30 REPeat star ↵
40   LET couleur = RND(6) ↵
50   IF couleur = 6 THEN EXIT star ↵
60   INK couleur ↵
70   LINE 50,60 TO RND(100),RND(100) ↵
80 END REPeat star ↵
```

La chose importante à noter ici est que le programme continue jusqu'à ce que la couleur devienne code 6. Il faut alors poursuivre hors de la boucle, juste après la ligne 90. Ici, il n'y a pas de ligne de programmes après la ligne 90, le programme s'arrête.

Un autre concept important a été évoqué. C'est l'idée de décision :

```
IF couleur = 6 THEN EXIT star
```

C'est une autre structure, très utile, car elle permet de faire un choix entre faire quelque chose ou non ; nous l'appellerons une décision binaire. Sa forme générale est :

```
IF condition THEN intruction(s)
```

Vous verrez plus tard comment les deux concepts de boucles et de décisions sont les principales structures pour contrôler la sortie d'une boucle. Vous pouvez aussi arrêter le programme en faisant un break : appuyez sur la touche **CTRL** puis sur **ESPACE**.

TEST SUR LE CHAPITRE 3

Vous pouvez gagner un maximum de 12 points dans le test suivant. Vérifiez votre score avec les réponses de la page suivante :

1. Qu'est-ce qu'un pixel ?

2. Combien y a-t-il de pixels dans la longueur de l'écran en mode basse résolution ?

3. Combien y a-t-il de pixels du bas en haut de l'écran en mode basse résolution ?

4. Quels sont les deux nombres qui déterminent « l'adresse » ou la position d'un point à l'écran, si on utilise une échelle ?

5. Combien de couleurs sont disponibles en mode basse résolution ?

6. Nommer les mots-clés qui exécutent ce qui suit :
   - choisir une échelle pour les tracés
   - tracer une droite
   - choisir une couleur pour le tracé
   - choisir une couleur pour le fond
   - tracer une bordure.

7. Quels sont les instructions qui ouvrent et ferment la boucle **REPEAT** ?

8. Pourquoi les boucles ont-elles des noms en SuperBASIC ?

REPONSES AU TEST DU CHAPITRE 3

1. Un pixel est la plus petite surface de lumière qui peut apparaître à l'écran.

2. Il a 256 pixels dans la longueur de l'écran, en basse résolution.

3. Il y a 256 pixels du bas en haut de l'écran en basse résolution.

4. Une adresse est déterminée par :
   - l'ordonnée de 0 à 100,
   - l'abscisse, de 0 à un nombre calculé par le système.

5. Il y a 8 couleurs disponibles à l'écran, en comptant le blanc et le noir.

6. Les réponses sont les suivantes :
   - SCALE sélectionne une échelle. Exemple : SCALE 100,0,0
   - LINE traçait une droite. Exemple : LINE a,b TO x,y
   - INK sélectionne une couleur pour les tracés. Exemple : INK 5
   - PAPER sélectionne la couleur du fond. Exemple : PAPER 7
   - BORDER trace une bordure. Exemple :BORDER 1,5

7. **REPeat** nom … **END REPeat** nom.

8. Une boucle **REPeat** se termine quand une instruction « **EXIT** nom » est exécutée.

9. En SuperBASIC, les boucles ont un nom pour qu'il soit possible d'en sortir pour revenir dans le programme lui-même.

Vérifier votre score :

11-12 : Très bien. Continuez à lire.

8-10 : Bien, mais relisez quelques parties du chapitre 3.

6-7 : Convenable, mais relisez quelques parties du chapitre 3 puis refaites le test.

0-5 : vous devez revoir attentivement le chapitre 3 et refaire le test.

PROBLEMES SUR LE CHAPITRE 3

1. Ecrivez un programme qui trace des lignes droites sur tout l'écran. Les droites doivent être de longueur et de direction aléatoires. Le début de chacune doit être la fin de la précédente, et chacune doit avoir une couleur choisie de façon aléatoire.

2. Écrire un programme qui trace des droites aléatoires avec la restriction que chaque droite a un point de départ aléatoire sur la bordure gauche de l'écran.

3. Écrire un programme qui trace des droites aléatoires avec la restriction que les droites partent d'un même point au bas de la bordure d'écran.

4. Écrire un programme qui construit des droites de longueurs, de points de départ et de couleurs aléatoires. Toutes les droites doivent être horizontales.

5. Écrire un programme qui construit une spirale carrée, de sorte que chaque partie soit de couleur aléatoire. Remarque : trouver d'abord les coordonnées des quatre sommets puis les répartir dans des groupes. Vous devez découvrir une règle.

CHAPITRE 4

CARACTERES ET CHAINES

Certains professeurs désirs parfois mesurer l'effort nécessaire à la lecture de certains livres. Il existe pour cela des tests variés et quelques-uns calculent longueur moyenne des mots et des phrases. Nous allons introduire quelques notions au sujet des manipulations de mots et de chaînes de caractères par quelques exemples, pour trouver la longueur moyenne d'un mot.

Nous parlons de suites de lettres, nombre autres symboles qui peuvent constituer ou non des mots. C'est pourquoi nous employons l'expression « chaîne de caractères ». Elle est habituellement abrégée en « chaîne ». Les chaînes sont manipulées de la même façon que les nombres, mais actuellement, nous ne pouvons pas leur appliquer les mêmes opérations. On ne multiplie ni ne soustrait des chaînes. On les assemble, on les partage, on fait une recherche à l'intérieur, et on les manipule généralement comme on le veut.

NOMS DES CHAINES ET CASES MEMOIRES

Vous pouvez créer des cases mémoire pour des chaînes. Vous pouvez placer des chaînes de caractères dans des cases et utiliser l'information exactement comme avec des nombres. Si vous avez l'intention de mettre en mémoire (mais pas tout de suite) des mots comme :

PREMIER SECOND TROISIEME
et
JANVIER FEVRIER MARS

vous pouvez choisir de nommer deux cases :

Noter la présence du signe $. À l'intérieur de la machine, les cases mémoires pour les chaînes sont différentes de celles utilisées par les nombres et SuperBASIC a besoin de savoir de quel type de cases mémoires il s'agit. C'est pourquoi tous les noms des cases mémoires des chaînes seront terminés par le signe $. Par ailleurs, les règles de choix des noms sont les mêmes que celles des noms des cases mémoires numériques.

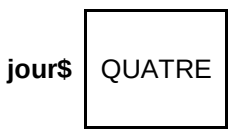
Vous devez prononcer :

jour$	jourdollar
mois$	moisdollar

L'instruction LET s'applique de la même manière qu'avec les nombres. Si vous tapez :

```
LET jour$ = "QUATRE" ↵
```

une case mémoires, nommée jour$ sera construite et QUATRE sera placé dedans :



Les guillemets ne sont pas chargés dans la case mémoire. Ils sont utilisés dans l'instruction LET pour que l'on sache clairement ce qui doit être chargé dans la case mémoire. Vous pouvez utiliser des apostrophes à la place des guillemets. Vous pouvez le vérifier en tapant :

```
PRINT jour$ ↵
```

et l'écran affichera le contenu de la case mémoire :

```
QUATRE
```

LONGUEUR DES CHAINES

SuperBASIC évalue aisément la longueur ou le nombre de caractères dans n'importe quelle chaîne. Vous devez simplement écrire :

```
PRINT LEN(jour$) ↵
```

Si la case mémoire jour$ contient « quatre » le nombre 6 sera affiché. Vous pouvez l'observer dans le petit programme suivant :

```
NEW ↵
10 LET jour$ = "QUATRE" ↵
20 PRINT LEN(jour$) ↵
RUN ↵
```

L'écran affichera :

```
6
```

LEN est un autre mot-clé du SuperBASIC.

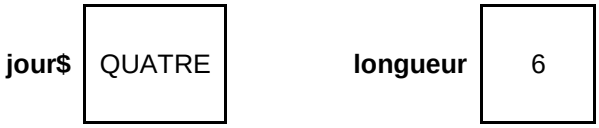
Une autre méthode consiste à utiliser à la fois une case mémoire pour la chaîne de caractères, et une autre, numérique :

```
NEW ↵
10 LET jour$ = "QUATRE" ↵
20 LET longueur = LEN(jour$) ↵
30 PRINT longueur ↵
RUN ↵
```

L'écran affichera :

```
6
```

comme précédemment, et de cases mémoires contiendront les valeurs indiquées :



Revenons maintenant au problème de longueur moyenne des mots. Écrivons un programme qui calcule la longueur moyenne de trois mots :

LE, QUATRE, FEVRIER

Si un problème vous paraît un peu élaboré, c'est une bonne idée de le décomposer avant d'écrire le programme lui-même.

1. Charger les trois mots dans des cases mémoires.
2. Calculer leur longueur et les charger.
3. Calculer la moyenne.
4. Afficher le résultat.

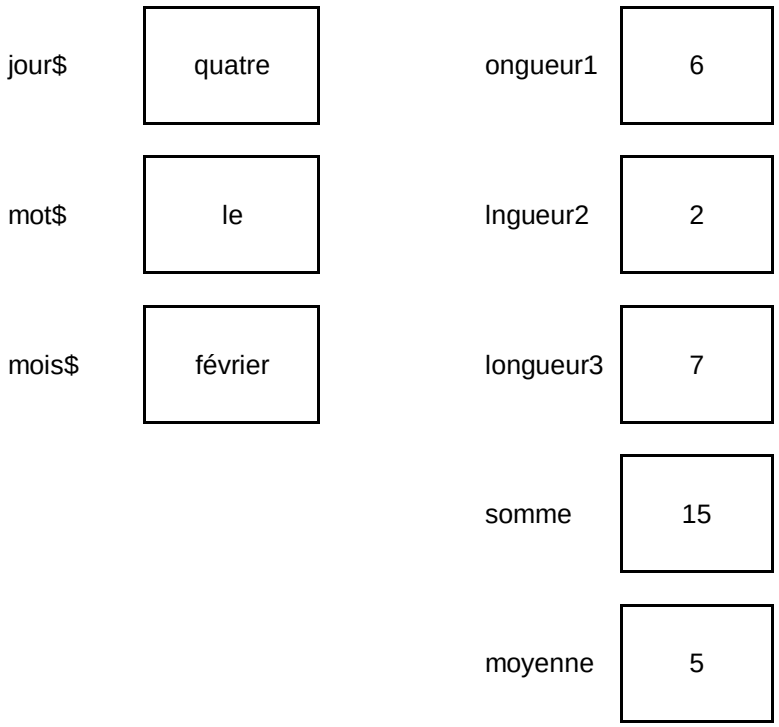
Le programme correspondant est le suivant :

```
NEW ↵
10 LET mot$ = "le" ↵
20 LET jour$ = "QUATRE" ↵
30 LET mois$ = "février" ↵
40 LET longueur1 = LEN (mot$) ↵
50 LET longueur2 = LEN (jour$) ↵
60 LET longueur3 = LEN (mois$) ↵
70 LET somme = longueur1 + longueur2 + longueur3 ↵
80 LET moyenne = somme/3 ↵
90 PRINT moyenne ↵
RUN ↵
```

Le symbole / signifie «divisé par». Le résultat final qui sera affiché sera :

```
5
```

et il y a 8 cases mémoires utilisées :



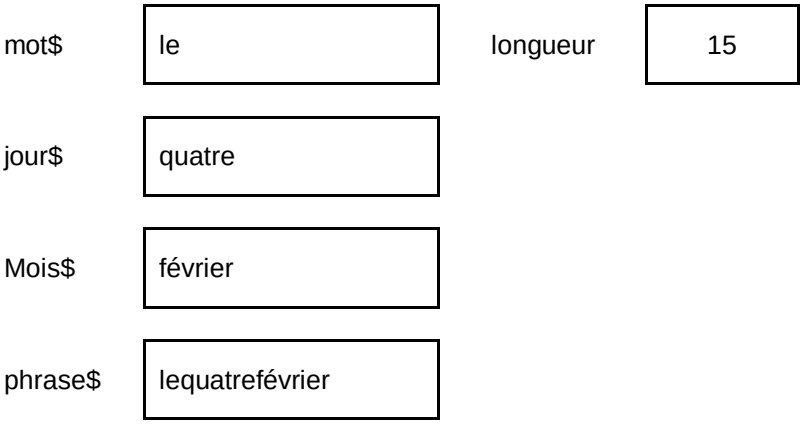
Si vous trouvez cela trop long pour un problème si simple, vous pouvez certainement le raccourcir. La version la plus courte ne contiendrait qu'une seule ligne, mais il serait assez difficile de la comprendre. Un compromis raisonnable utilise le symbole & qui permet l'opération :

concaténation de 2 chaînes

Tapez maintenant :

```
NEW ↵
10 LET mot$ = "le" ↵
20 LET jour$ = "quatre" ↵
30 LET mois$ = "février" ↵
40 LET phrase$ = mot$ & jour$ & mois$ ↵
50 LET longueur = LEN(phrase$) ↵
60 PRINT longueur/3 ↵
RUN ↵
```

Le résultat affiché est 5 comme précédemment, mais il y a quelques différences internes :



Il y a encore une simplification possible qui est d'utiliser **READ** et **DATA** au lieu des trois premières instructions **LET**. Tapez :

```
NEW ↵
5 RESTORE : CLS ↵
10 READ mot$, jour$, mois$ ↵
20 LET phrase$ = mots$ & jour$ & mois$ ↵
30 LET longueur = LEN(phrase$) ↵
40 PRINT longueur/3 ↵
50 DATA "le","quatre","février" ↵
RUN ↵
```

Le résultat interne de cette version est exactement le même que celui de la version précédente. **READ** provoque la création et le chargement de valeur de cases mémoires de la même manière que **LET**.

IDENTIFIANTS ET VARIABLES CHAINES

Les noms des cases mémoires tels que :

mot$
jour$
mois$
phrase$

sont appelés des identifiants de chaînes. Le dollar implique que les cases mémoires contiennent des chaînes de caractères. Dollar doit toujours être en dernier. On prononce de tels mots comme motdollar, jourdollar etc…

Les cases mémoires de cette sorte sont appelées des variables chaînes parce qu'elles ne contiennent que des chaînes de caractères qui peuvent varier en cours d'exécution du programme

Les contenus de telles cases mémoires sont appelés valeurs. Ainsi des mots tels que « le » et « quatre » peuvent être les valeurs des variables chaînes nommées mot$ et jour$.

CARACTERES ALEATOIRES

Vous pouvez utiliser les codes de caractères (voir le concept de Référence Guide) pour générer des lettres aléatoires. Les lettres majuscules de A à Z ont les codes de 65 à 90. La fonction CHR$ convertit ces codes en lettres. Le programme suivant va imprimer la lettre B.

```
NEW ↵
10 LET codelettre = 66 ↵
20 PRINT CHR$ (codelettre) ↵
RUN ↵
```

Le programme suivant génère des triplets de lettres A, B, C jusqu'à ce que le mot CAB soit accidentellement épelé.

```
NEW ↵
10 REPeat taxi
20 LET prem$ = CHR$(RND(65 TO 67))
30 LET second$ = CHR$(RND(65 TO 67))
40 LET trois$ = CHR$(RND(65 TO 67))
50 LET mot$ = prem$ & second$ & trois$
60 PRINT ! mot$ !
70 IF mot$ = "CAB" THEN EXIT taxi
80 END REPeat taxi
```

Les caractères aléatoires, comme les nombres et les points aléatoires sont utilisés pour apprendre à programmer. Vous pouvez facilement obtenir des résultats intéressants dans des exemples de programmes ou exercices.

TEST SUR LE CHAPITRE 4

Vous pouvez gagner un maximum de 10 points pour ce test. Calculez votre score avec les réponses de la page suivante.

1. Qu'est-ce qu'une chaîne de caractères ?
2. Quel est l'abréviation usuelle du terme « chaîne de caractères » ?
3. Qu'est-ce qui distingue que le nom d'une chaîne de caractères ?
4. Comment prononce-ton : « mot$ » ?
5. Quel mot-clé est utilisé pour chercher le nombre de caractères d'une chaîne ?
6. Quel symbole est utilisé pour concaténer les deux chaînes ?
7. Les espaces peuvent être contenus dans une chaîne. Comment sont définies les limites d'une chaîne ?
8. Quand une instruction telle que :
   ```
   LET viande$ = "steak"
   ```
   est exécutée, est-ce que les guillemets sont chargés ?
9. Quelle fonction peut transformer un nombre codé en lettre ?
10. Comment pouvez-vous générer des lettres majuscules aléatoires ?

REPONSES AU TEST DU CHAPITRE 4

1. Une chaîne de caractères est une suite de caractères tels des lettres, des nombres ou d'autres symboles.

2. L'expression « chaîne de caractères » est souvent abrégée en « chaîne ».

3. Le nom d'une variable chaîne se termine toujours par le caractère $.

4. Les mots quelques mot$ sont prononcés motdollar.

5. C'est le mot-clé LEN qui trouvera la longueur ou le nombre de caractères d'une chaîne. Par exemple, si la variable viande$ à la valeur « steak » alors l'instruction :
   ```
   PRINT LEN (viande$)
   ```
   donnera en sortie : 5

6. le symbole pour concaténer les deux chaînes est &.

7. Les limites d'une chaîne peuvent être définies par des guillemets ou des apostrophes.

8. Les guillemets ne font pas partie de la chaîne et ne sont pas chargés.

9. La fonction est **CHR$**. Elle est utilisée avec des parenthèses comme dans CHR$(66) ou CHR$(RND(65 TO 67))

10. On génère des lettres aléatoires avec les instructions :
    ```
    cod = RND(65 TO 90)
    PRINT CHR$(cod)
    ```

Calculez votre score :

9 ou 10 : Très bien. Continuez à lire.

7-8 : Bien, mais relisez quelques parties du chapitre 4.

5 ou 6 : Convenable, mais relisez quelques parties du chapitre 4 puis refaites le test.

0-5 : vous devez revoir attentivement le chapitre 4 et refaire le test.

CHAPITRE 5

BONNE PRATIQUE DEJA ACQUISE

Vous avez déjà commencé à travailler de manière effective sur de petits programmes. Vous avez peut-être remarqué que les consignes suivantes ne sont pas inutiles :

1. Utilisation de minuscules pour les identifiants : noms de variable (cases mémoires) ou structures **REPeat** etc…

2. Emploi des modules qui montrent le contenu d'une structure **REPeat**.

3. Bien choisir les identifiants, de manière qu'il reflète ce qu'exprime une variable ou une structure REPeat.

4. Modification d'un programme en :
   - remplaçant une ligne,
   - insérant une ligne,
   - supprimant une ligne.

EXEMPLE DE PROGRAMMES

Vous avez atteint le stade où il est bon d'être capable d'étudier des programmes pour essayer de comprendre ce qu'ils font, et comment ils le font. Le mécanisme de leur déroulement devrait maintenant être bien compris, et dans les chapitres suivants nous nous dispenserons de la répétition constante de :

NEW avant chaque programme

↵ à la fin de chaque ligne

RUN pour démarrer l'exécution de chaque programme

Vous saurez qu'il faut utiliser ces instructions, mais leur omission dans le texte vous permettra de voir plus clairement les autres détails.

Si nous omettons les détails ci-dessus, nous pouvons utiliser et comprendre les programmes plus facilement censés surcharges techniques. Par exemple, le programme suivant génère des lettres majuscules jusqu'à l'apparition de la lettre « Z ». Les mots et les signes NEW, ↵ P RUN ne figurent pas mais vous devez encore les utiliser :

```
10 REPeat lettres
20 LET codelettre = RND(65 TO 90)
30 cap$ = CHR$(codelettre)
40 PRINT cap$
50 IF cap$ = "Z" THEN EXIT lettres
60 END REPeat lettres
```

Dans ce chapitre les suivants, les programmes seront indiqués sont les symboles d'ENTREE. Les commandes directes aussi seront indiquées sont les symboles ENTREE. Mais vous devez utiliser ces touches comme d'habitude. Vous devez aussi vous rappeler d'utiliser **NEW** et **RUN**.

NUMEROTATION AUTOMATIQUE DES LIGNES

Il est fastidieux d'entrer les numéros de lignes à la main. Au lieu de cela vous pouvez taper :

```
AUTO
```

et alors, le QL répondra par un numéro de ligne :

```
100
```

Continuez à taper jusqu'à la fin de votre programme :

```
100 PRINT "Premier"
110 PRINT "Deuxième"
120 PRINT "Fin"
```

Pour terminer la numérotation automatique, utilisez un break : **CTRL** et **ESPACE**. Ceci produira le message :

```
130 not complete
```

et la ligne 130 ne sera pas incluse dans votre programme.

Si vous faites une erreur qui ne justifie pas un break de la numérotation automatique, vous pouvez continuer et éditer la ligne plus tard. Si vous faites une erreur de syntaxe, un break survient, suivi d'un message d'erreur. Si vous voulez alors continuer à la ligne 110, tapez :

```
AUTO 110
```

et continuez comme précédemment.

Si vous désirez démarrer un numéro de ligne particulier, et utiliser un incrément autre que 10, vous pouvez écrire, par exemple pour le numéro de ligne 600 et l'incrément 5 :

```
AUTO 600,5
```

Les lignes seront alors numérotées : 600, 605, 610, etc.

ÉDITION D'UNE LIGNE

Pour éditer une ligne, il suffit de taper **EDIT** suivi du numéro de ligne, par exemple :

```
EDIT 100
```

La ligne sera alors affichée avec le curseur à la fin, ainsi :

```
110 PRINT "Premier"
```

Vous pouvez utiliser le curseur en utilisant les touches :

- ← un caractère vers la gauche
- → un caractère vers la droite

Pour supprimer un caractère à gauche :

```
CTRL plus la touche ←
```

Pour supprimer un caractère situé sous le curseur, tapez :

```
CTRL plus la touche □
```

et le caractère à la droite du curseur se rapprochera pour éviter un espace.

UTILISATION DES MICRODRIVES

Avant d'utiliser une nouvelle cartouche microdrive, il faut la formater. Suivez les instructions du « guide d'introduction QL ». Le choix d'un nom pour la cartouche est soumis aux mêmes règles que celui des identifiants SuperBASIC, mais il doit être limité à 10 caractères. Il est bon d'écrire le nom de la cartouche sur la cartouche elle-même, en utilisant une des étiquettes fournies.

Vous devez toujours garder au moins une copie (sur cartouche) de tous les programmes ou données, comme l'indiquent les instructions de la partie Information située à la fin du guide Utilisateur.

ATTENTION

Si vous formatez une cartouche qui contient des programmes et/ou des données, tous les programmes et les données seront perdus.

SAUVEGARDE DES PROGRAMMES

Le programme suivant construit des bordures, de 8 pixels de large, en rouge (code 2), dans trois fenêtres désignées #0, #1, #2.

```
100 REMark Bordures
110 FOR k = 0 TO 2 : BORDER #k,8,2
```

Vous pouvez sauvegarder ce programme sur un microdrive en insérant une cartouche et en tapant :

```
SAVE mdv1_bord
```

Le programme sera sauvegardé dans un fichier microdrive appelé bord.

REPERTOIRE D'UNE CARTOUCHE

Si vous voulez savoir quels programmes ou quels fichiers sont sur une cartouche particulière, il faut la placer dans le microdrive 1 et taper :

```
DIR mdv1_
```

le répertoire sera affiché à l'écran.

Le répertoire sera affiché à l'écran. Si la cartouche est dans le Microdrive 2 alors, tapez :

```
DIR mdv2_
```

COPIES DE PROGRAMMES ET DE FICHIERS

Une fois qu'un programme est chargé comme un fichier sur une cartouche Microdrive, il peut être copié dans d'autres fichiers. On peut aussi faire la copie d'une cartouche Microdrive. Vous pourriez copier tous les programmes existant sur une autre cartouche en tapant :

```
COPY mdv1_bord TO mdv2_bord
```

SUPPRESSION D'UN FICHIER D'UNE CARTOUCHE

Un fichier c'est quelque chose qui contient un programme ou des données, chargés sur une cartouche. Pour supprimer un programme appelé prog vous tapez :

```
DELETE mdv1_prog
```

CHARGEMENT DE PROGRAMMES EN MEMOIRE

Un programme peut être chargé à partir d'une cartouche dans un Microdrive en tapant (en supposant que le programme à charger se nomme « Bord ») :

```
LOAD mdv2_bord
```

Si le programme s'est chargé correctement, alors vous en avez deux exemplaires. Vous pouvez tester le programme que vous venez de charger en tapant :

`LIST` pour afficher à l'écran
`RUN` pour l'exécuter.

Au lieu d'utiliser LOAD suivi de RUN, vous pouvez combiner les deux opérations en une seule commande :

```
LRUN mdv2_bord
```

Le programme se chargera et s'exécutera immédiatement.

FUSION DE PROGRAMMES

Supposons que vous ayez deux programmes sauvegardés sur microdrive 1 et nommés prog1 et prog2 :

Prog1 : `10 PRINT "premier"`

Prog2 : `20 PRINT "second"`

Si vous tapez :

```
LOAD mdv1_prog1
```

suivi de :

```
MERGE mdv1_prog2
```

les deux programmes seront fusionnés en un seul. Pour le vérifier, tapez LIST et vous verrez :

```
10 PRINT "premier"
10 PRINT "second"
```

Si vous fusionnez un programme à un autre, assurez-vous que tous ses numéros de ligne sont différents de ceux du programme qui est déjà en mémoire centrale. Sinon il y aurait dans le premier programme des lignes qui seraient écrasées. Cette commodité est très appréciable pour ceux qui utiliseront souvent cette procédure. Il devient alors facile de construire un programme en lui ajoutant des procédures et des fonctions.

REMARQUES GENERALES

Dans l'emploi des cartouches, soyez toujours attentif et méthodique. Gardez toujours une sauvegarde, et si vous craignez un problème quelconque avec une cartouche ou un microdrive, gardez une seconde sauvegarde. Les professionnels de l'informatique perdent très rarement leurs données. Ils savent que même les meilleures machines ou périphériques peuvent avoir des problèmes occasionnels, et ils prennent des précautions.

Si vous voulez appeler un programme par un nom particulier, comme "carré", il est bon d'utiliser des noms comme car1, car2 pour les versions préliminaires. Quand le programme est dans sa forme définitive, vous pouvez alors en faire deux copies appelées "carré" et supprimer les autres en reformatant la cartouche ou par d'autres méthodes.

T E S T SUR LE CHAPITRE 5

Vous pouvez gagner un maximum de 13 points dans le test suivant. Calculez votre score à l'aide des réponses de la page suivante.

1. Pourquoi les lettres minuscules sont-elles préférables pour les mots du programme que vous choisissez vous-même ?

2. Quelle est la fonction d'un module ?

3. Qu'est-ce qui doit normalement guider votre choix d'identifiants pour les variables et les boucles ?

4. Indiquer trois façons de modifier un programme chargé en mémoire centrale de l'ordinateur. (3 pts)

5. Quelle instruction ne devez-vous pas oublier à la fin de chaque commande ou de chaque ligne de programme que vous entrez ?

6. Que devez-vous normalement taper avant d'entrer un programme au clavier ?

7. Que faut-il écrire au début de chaque ligne qui doit être chargée comme une partie de programme ?

8. Quelle instruction faut-il taper pour obtenir l'exécution d'un programme ?

9. Quel est le mot-clé qui vous permet d'écrire des informations dans un programme, et qui n'a aucun effet à l'exécution ?

10. Quels sont les mots-clés qui vous permettent de charger les programmes sur les cartouches, et de les retrouver ? (2 pts)

REPONSES AU TEST DU CHAPITRE 5

1. Les lettres minuscules pour les noms de variables ou de boucles se distinguent des mots-clés qui sont en partie affichés en majuscules.

2. Les modules révèlent clairement le contenu des boucles (ou autres structures).

3. Les identifiants devraient normalement être choisis de manière à exprimer quelque chose, par exemple "compte" ou "mot$" plutôt que C ou M$.

4. Vous pouvez modifier un programme chargé en :
   - remplaçant une ligne,
   - insérant une ligne,
   - supprimant une ligne.

5. Pour entrer une commande ou une ligne de programme, il faut utiliser la touche ENTREE.

6. Le mot NEW doit éliminer le précédent programme SuperBASIC du QL et doit vous assurer que le nouveau programme que nous allons entrer ne sera pas fusionné avec l'ancien.

7. Si vous souhaitez qu'une ligne soit chargée comme une partie de programme, vous devez alors indiquer un numéro de ligne.

8. Pour obtenir l'exécution d'un programme, il faut taper **RUN**.

9. Le mot **REMark** permet d'entrer des informations dans un programme, sans qu'elles soient prises en compte à l'exécution.

10. Les mots-clés **SAVE** et **LOAD** permettent de copier des programmes sur la cartouche, et de les charger en mémoire centrale.

Calculez votre score :

12-13 : Très bien - Continuez à lire.

10-11 : Bien, mais relisez quelques parties du chapitre 5.

8 – 9 : Convenable, mais relisez quelques parties du chapitre 5 et refaites le test.

0 – 8 : Vous devriez relire attentivement le chapitre 5 et refaire le test.

EXERCICES SUR LE CHAPITRE 5.

1. Réécrire le programme suivant en utilisant des lettres minuscules pour donner une meilleure présentation. Ajouter les mots **NEW** et **RUN**. Indiquer des numéros de lignes et le mot clé **ENTREE** où vous voulez pour entrer et exécuter un programme. Utilisez le mot-clé **REMark** pour donner un nom au programme.

```
LET DEUX$ = "DEUX"
LET QUATRE$ = "QUATRE"
LET SIX$ = DEUX$ & QUATRE$
PRINT LEN(six$)
```

   Expliquer comment deux et quatre peuvent faire dix.

2. Utilisez la forme modulaire, les lettres minuscules, **NEW**, **RUN**, les numéros de ligne et **ENTREE** pour entrer et exécuter le programme suivant

```
REPEAT LOOP
LETTRE_CODE = RND(65 TO 90)
LET LETTRES$ = CHR$(LETTRE_CODE)
PRINT LETTRE$
IF LETTRE$ = 'Z' THEN EXIT LOOP
END REPEAT LOOP
```

3. Réécrire le programme suivant dans un meilleur style, utilisant des noms de variables expressifs et ayant une bonne présentation.

```
LET S = O
REPeat TOTAL
LET N = RND(1 TO 6)
PRINT ! N !
LET S = S + N
IF n = 6 THEN EXIT TOTAL
END REPeat TOTAL
PRINT S
```

   A quoi sert ce programme ? Entrez-le et exécutez-le pour vérifier ce que vous avez trouvé.

CHAPITRE 6

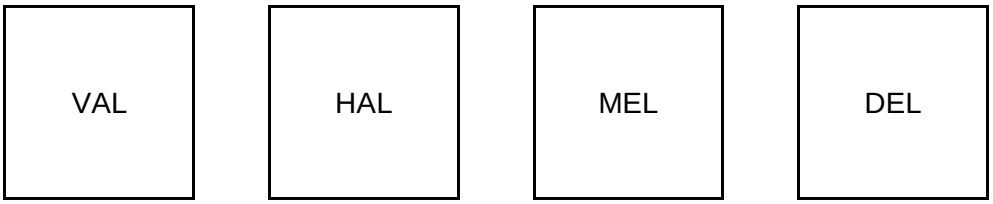
TABLEAUX ET BOUCLES FOR

QU'EST-CE QU'UN TABLEAU ?

Vous savez que les nombres ou les chaînes de caractères peuvent être considérés comme des valeurs de variables. Vous pouvez les représenter comme des nombres ou des mots contenus dans des cases mémoire ou des maisons. Supposons, par exemple, que quatre employés d'une compagnie sont envoyés dans un petit village, peut-être parce qu'on y a trouvé du pétrole. Dans ce village, il y a quatre maisons à louer. Les noms de toutes les maisons se terminent par le symbole $.

Westlea$ Lakeside$ Roselawn$ Oaktree$

Les quatre employés sont appelés :



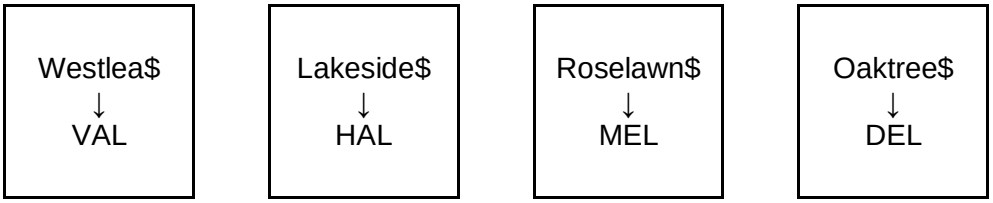
Ils peuvent être placés dans les maisons par deux méthodes.

Programme 1.

```
100 LET westlea$ = "VAL"
110 LET lakeside$ = "HAL"
120 LET roselawn$ = "MEL"
130 LET oaktree$ = "DEL"
140 PRINT ! westlea$ ! lakeside$ ! roselawn$ ! oaktree$
```

Programme 2.

```
10 RESTORE
100 READ westlea$, lakeside$, roselawn$, oaktree$
110 PRINT ! westlea$ ! lakeside$ ! roselawn$ ! oaktree$
120 DATA "VAL", "HAL", "MEL", "DEL"
```



Quand la quantité de données augmente, les avantages de **READ** et **DATA** sur **LET** sont plus importants. Mais quand les données deviennent réellement nombreuses, le problème de trouver un nom pour chaque maison devient aussi difficile que celui de trouver des maisons vacantes dans un tout petit village.

La solution à ce problème, et à d'autres, réside dans un nouveau type de case mémoire dans laquelle plusieurs éléments peuvent prendre le même nom. Cependant, ils doivent être distincts, c'est pourquoi chaque variable aura toujours un numéro, comme les maisons numérotées d'une même rue. Supposons qu'il vous faut quatre maisons libres dans High Street, numérotées de 1 à 4.

En SuperBASIC nous parlerons d'un **tableau** de quatre maisons. Le nom de ce tableau est *high_st$* et les quatre maisons sont numérotées de 1 à 4.

Mais vous ne pouvez pas utiliser ce tableau de variables comme de simples variables. Vous devez d'abord déclarer la taille du tableau. L'ordinateur réserve l'emplacement interne nécessaire et il lui faut connaître le nombre de variables chaînes du tableau, et aussi la longueur maximale de chaque variable chaîne. On utilise l'instruction **DIM** de la façon suivante :

```
DIM high_st$(4, 3)
```

longueur maximale d'une chaîne

nombre de variables chaînes

Après l'exécution de l'instructin **DIM**, les variables sont disponibles. C'est comme si les maisons avaient été construites, mais étaient encore vides. Les quatre « maisons » ont le même nom, mais chacune a son propre numéro et chacune peut contenir jusqu'à 3 caractères.

Les cinq programmes ci-dessous font tous la même chose : ils provoquent la création et le chargement des quatre « maisons » et ils affichent leur contenu. La méthode finale n'utilise que quatre lignes, mais les quatre centres vous conduisent vers le but, en partant du connu vers l'inconnu. Le mouvement va toujours vers une plus grande économie d'instructions.

Si vous comprenez parfaitement les deux ou trois premières méthodes, vous pouvez travailler directement les méthodes 4 et 5. Mais si vous avez un doute, les méthodes 1, 2 et 3 vous aideront à éclaircir les choses.

Programme 1

```
100 DIM high_st$(4,3)
110 LET high_st$(l) = "VAL"
120 LET high_st$(2) = "HAL"
130 LET high_st$(3) = "MEL"
140 LET high st$(4) = "DEL"
150 PRINT ! high_st$(1) ! high st$(2) !
160 PRINT ! high_st$(3) ! high-st$(4) !
```

Programme 2

```
100 DIM high st$(4,3)
110 READ high_st$(1),high_st$(2),high_st$(3),high_st$(4)
120 PRINT ! high_st$(1) ! high_st$(2) !
130 PRINT ! high_st$(3) ! high_st(4) !
140 DATA "VAL","HAL","MEL","DEL"
```

Ceci montre comment économiser des noms de variables, mais la constante répétition de *high_st$* est fastidieuse et donne aux programmes une certaine lourdeur. Nous pouvons aussi utiliser une technique comme la boucle **REPeat**, pour améliorer certains points. Nous utilisons un compteur, nombre qui augmente de 1 à chaque passage dans la boucle.

Programme 3

```
100 RESTORE 190
110 DIM high_st$(4,3)
120 LET nombre = 0
130 REPeat maisons
140   LET nombre = nombre + 1
150   READ high_st$(nombre)
160   IF num = 4 THEN EXIT maisons
170 END REPeat maisons
180 PRINT high_st$(1) ! high_st$(2) ! high_st$(3) ! high_st$(4)
190 DATA "VAL","HAL","MEL","DEL"
```

Ce type de boucle, dans lequel on fait quelque chose un certain nombre de fois, est bien connu. Une structure spéciale, appelée boucle **FOR**, a été conçue pour cela. Dans une telle boucle, le compte de 1 à 4 s'effectue automatiquement. La sortie a lieu quand les quatre passages sont exécutés.

Programme 4

```
100 RESTORE 160
110 DIM high_st$(4,3)
120 FOR nombre = 1 TO 4
130 READ high_st$(nombre)
140 PRINT ! high_st$(nombre) !
150 END FOR nombre
160 DATA "VAL","HAL","MEL","DEL"
```

La sortie pour les quatre programmes est la même :

```
VAL HAL MEL DEL
```

montrant que les données sont correctement chargées dans le tableau de quatre variables.

La méthode 4 est, sans aucun doute, la meilleure car elle s'applique de la même façon pour 4, 40 ou 400 articles, en changeant seulement le nombre 4 et en ajoutant d'autres noms en **DATA**. Vous pouvez utiliser autant d'instructions **DATA** que vous désirez.

Dans sa plus simple forme, la boucle **FOR** est semblable à la plus simple forme de la boucle **REPeat**. On peut comparer les deux :

```
100 REPeat saluation          100 FOR salutation = 1 TO 40
110 PRINT "Bonjour"           110 PRINT "Bonjour"
120 END REPeat salutation     120 END FOR salutation
```

Ces boucles fonctionneraient toutes les deux. La boucle **REPeat** imprimerait 'Bonjour' sans fin (il faudrait l'arrêter par un break » et la boucle **FOR** imprimerait 'Bonjour' juste 40 fois.

Il faut noter que le nom de la boucle **FOR** est aussi une variable, salutation, dont la valeur varie de 1 à 40 dans le déroulement du programme. Cette variable est quelquefois appelée la **variable boucle**, ou la **variable de contrôle de la boucle**.

Notez aussi que la structure des deux boucles prend la forme :

Instruction d'ouverture
Contenu
Instruction de fermeture

Cependant certaines structures admettent une forme courte qui peut être utilisée lorsque la boucle contient une seule instruction, ou un petit nombre. La forme courte de la boucle **FOR** se traduirait de la manière suivante, permettant d'écrire le programme de la manière la plus économique :

Programme 5 :

```
100 RESTORE 140 : CLS
110 DIM high_st$(4,3)
120 FOR nombre = 1 TO 4 : READ high_st$(nombre)
130 FOR nombre = 1 TO 4 : PRINT ! high_st$(nombre) !
140 DATA "VAL", "HAL", "MEL", "DEL"
```

Les deux points servent de symboles de fin d'instruction et remplacent **ENTREE**, et le symbole **ENTREE** des lignes 20 et 30 sert d'instruction **END FOR**.

Il y a même une autre manière plus courte d'écrire le programme 5. Pour éditer le contenu du tableau *high_st$* nous pouvons remplacer la ligne 130 par :

```
130 PRINT ! high_st$ !
```

Ceci utilise une tranche de tableau dont nous parlerons plus tard au chapitre 13.

Nous avons introduit le concept de tableaux de variables de chaînes, mais les tableaux peuvent contenir des chaînes ou des nombres, et l'exemple suivant illustre bien un tableau numérique.

Imaginons que l'on jette deux dés (comme au jeu de Monopoly) 400 fois. Garder le nombre d'occurrences de chaque score possible de 2 à 12.

```
100 REMark des1
110 LET deux = 0  :trois = 0:quatre = 0:cinq = 0:six = 0
120 LET sept = 0:huit = 0:neuf = 0:dix = 0 :onze = 0:douze = 0
130 FOR jeu = 1 TO 400
140   LET de1 = RND(1 TO 6)
150   LET de2 = RND(1 TO 6)
160   LET score = de1 + de2
170   IF score = 2 THEN LET deux = deux + 1
180   IF score = 3 THEN LET trois = trois + 1
190   IF score = 4 THEN LET quatre = quatre + 1
200   IF score = 5 THEN LET cinq = cinq + 1
21O   IF score = 6 THEN LET six = six + 1
220   IF score = 7 THEN LET sept = sept + 1
230   IF score = 8 THEN LET huit = huit + 1
240   IF score = 9 THEN LET neuf = neuf + 1
250   IF score = 10 THEN LET dix = dix + 1
26O   IF score = 11 THEN LET onze = onze + 1
270   IF score = 12 THEN LET douze = douze + 1
280 END FOR jeu
290 PRINT 'SCORE': FOR i=2 TO 12: PRINT I !!!
300 PRINT \'Coups'\ ! deux ! trois ! quatre ! cinq ! six ! sept ! huit !
neuf ! dix ! onze ! douze
```

Dans le programme ci-dessus, nous avons créé onze variables pour contenir le décompte des scores. Si vous représentez graphiquement les décomptes affichés en fin de programme, vous remarquez que la courbe représentative est sensiblement triangulaire. Les plus forts décomptes sont ceux des scores six, sept, huit, et les plus bas sont pour deux et onze. Comme chaque joueur de Monopoly l'a sans doute remarqué, ceci met en évidence la haute fréquence des scores du milieu, et la faible fréquence des deux et douze.

Programme 2 :

```
100 REMark Des2
110 DIM compte(12)
120 FOR jeu = 1 TO 400
130 LET de_1 = RND(1 TO 6)
140 LET de_2 = RND(1 TO 6)
150 LET score = de_1 + de_2
160 LET compte(score) = compte(score) + 1
170 END FOR jeu
180 FOR nombre = 2 TO 12 : PRINT ! compte(nombre) !
```

Dans la première boucle **FOR**, jeu, l'indice de la variable tableau est score. De ce fait, c'est le compte indicé par score qui sera incrémenté à chaque passage dans la boucle jeu. On peut comparer ce tableau à une suite de cases mémoires numérotées de 2 à 12. A chaque fois qu'un score particulier est porté, l'occurrence de ce score est incrémentée, et la case mémoire correspondante augmente de 1.

Dans la seconde boucle **FOR** (forme courte) l'indice est : nombre. Comme la valeur de nombre varie de 2 à 12, les différentes valeurs des comptes sont affichées.

Il faut noter que dans l'instruction **DIM** appliquée aux tableaux numériques, il vous suffit de déclarer le nombre de variables nécessaires. On ne précise pas la longueur maximum, comme c'est le cas dans un tableau de chaînes de caractères.

Si vous avez utilisé d'autres versions de BASIC, vous êtes sans doute étonné de ne pas rencontrer l'instruction **NEXT**. Mais toutes les structures en SuperBASIC doivent se terminer par **END** suivi du nom de la structure. Il est donc logique de n'avoir pas rencontré **NEXT** ; toutefois, il interviendra dans certains cas, comme vous le verrez dans les prochains chapitres.

TEST SUR LE CHAPITRE 6

Vous pouvez gagner un maximum de 16 points dans le test suivant. Calculez votre score en comparant vos résultats aux réponses de la page suivante.

1. Mentionner les deux types de difficultés rencontrées dans un programme qui contient de nombreuses données, si vous les traitez, sans tableau.

2. Si, dans un tableau, dix variables ont le même nom, alors comment les distinguez-vous ?

3. Quelle est l'instruction utilisée pour déclarer les éléments d'un même tableau ?

4. Quel est le terme utilisé pour distinguer une variable particulière d'un tableau ?

5. Quelle comparaison pouvez-vous faire entre deux éléments de la vie courante, et le concept de tableau en programmation ? (2 pts)

6. On sort d'une boucle **REPeat** lorsque la réalisation d'une condition provoque l'exécution de l'instruction **EXIT**. Comment sort-on d'une boucle **FOR** ?

7. Dans une boucle **REPeat**, il y a un nom qui permet de sortir correctement de la boucle. Une boucle **FOR** a aussi un nom, mais quelle est l'autre fonction de son nom ?

8. Quelles sont les deux expressions utilisées pour décrire la variable qui est le nom de la boucle **FOR** ? (2 Pts)

9. Les valeurs d'une variable boucle changent de façon automatique dans une boucle **FOR**. Indiquer une application possible de cette variation.

10. Les boucles **REPeat** et **FOR** (forme longue) ont des caractéristiques communes. Parmi les caractéristiques suivantes, lesquelles sont communes aux deux boucles, et lesquelles sont distinctes ?
    - Un mot-clé d'ouverture.
    - Un mot-clé de fermeture.
    - Un nom de boucle.
    - Une variable boucle ou une variable de contrôle. (4 pts)

REPONSES AU TEST SUR LE CHAPITRE 6

1. Il est difficile d'inventer un grand nombre de noms de variables pour le chargement de données. Et même si vous en inventez beaucoup, chacun devra être rentré avec une instruction **LET** ou **READ**, si vous n'utilisez pas de tableau.

2. C'est le nombre nommé indice qui distingue les variables. Toutes les variables du tableau partagent le même nom, mais chacune a un indice différent.

3. Il faut déclarer un tableau, en indiquant sa taille, avec l'aide d'une instruction **DIM**, placée généralement en début de programme.

4. Le terme utilisé pour distinguer une variable d'un tableau est appelé l'indice.

5. Les maisons d'une rue se partagent le même nom de rue, mais chacune a son propre numéro.

   Les lits d'une chambre d'hôpital peuvent se partager le nom de la chambre, mais chacun est numéroté.

   Dans une prison, plusieurs cellules peuvent appartenir au même bloc, elles ont donc le même nom de bloc, mais des numéros différents. (2 pts)

6. Une boucle **FOR** se termine quand le traitement correspondant à la dernière valeur de la variable boucle a été exécuté.

7. Le nom d'une boucle **FOR** est aussi le nom de la variable qui contrôle la boucle.

8. Les expressions utilisées pour la variable sont : variable boucle et variable de contrôle. (2 pts)

9. Les valeurs de la variable boucle peuvent être utilisées comme indices du nom de la variable tableau. Aussi, dans le déroulement de la boucle, chaque élément du tableau est « visité » une fois.

10. Les boucles **FOR** et **REPeat** ont toutes les deux :
    - un mot-clé d'ouverture.
      REPeat FOR
    - un mot clé de fermeture.
      END REPeat nom END FOR nom
    - un nom de boucle.
    - il n'y a que la boucle **FOR** qui possède une variable boucle et une variable de contrôle (4 pts)

Calculez votre score

Ce test est plus difficile que les précédents.

15 - 16 : Excellent. Continuez votre lecture.

13 - 14 : Très bien. Mais regardez tout de même comment procèdent quelques programmes.

11 - 12 : Bien, mais il faut mieux relire quelques passage du chapitre 6.

8 – 10 : Convenable, mais relisez quelques passages, et refaites le test.

0 – 7 : Relire attentivement le chapitre 6 et refaire le test.

EXERCICES SUR LE CHAPITRE 6

1. Utiliser une boucle **FOR** pour mettre l'un des quatre nombres 1, 2, 3, 4, de façon aléatoire, dans un tableau de 5 variables : card(1), card(2), card(3), card(4), card(5).

   Le même nombre peut être utilisé plusieurs fois.

   Utiliser une seconde boucle **FOR** pour afficher à l'écran les valeurs des cinq variables card.

2. Quatre nombres 1, 2, 3, 4 représentent les valeurs Cœur, Pique, Trèfle, Carreau. Quelles lignes de programme faut-il rajouter pour obtenir en sortie ces quatre mots au lieu des nombres ?

3. Utilisez une boucle **FOR** pour mettre cinq nombres aléatoires compris entre 1 et 13 dans un tableau de cinq variables :

   card(1), card(2), card(3), card(4), card(5).

   Utilisez une seconde boucle **FOR** pour obtenir en sortie les cinq variables card.

4. Imaginez que les nombres aléatoires générés dans le programme n° 3 représentent des cartes. Ajoutez les instructions qui permettent d'obtenir en sortie :

Nombre	**Sortie**
1	le mot "As"
2 to 10	le nombre lui-même
11	le mot "Valet"
12	le mot "Dame"
13	le mot "Roi"

CHAPITRE 7

PROCEDURES SIMPLES

MODULES

Si vous désirez écrire des programmes qui résolvent des problèmes complexes, vous pouvez remarquer qu'il est difficile d'en suivre le raisonnement jusqu'au bout. C'est pourquoi un esprit méthodique divise un travail long et complexe en plus petites parties ou tâches, puis il divise encore ces tâches en des tâches plus petites, et ainsi de suite, jusqu'à ce que chacune puisse être exécutée sans difficulté.

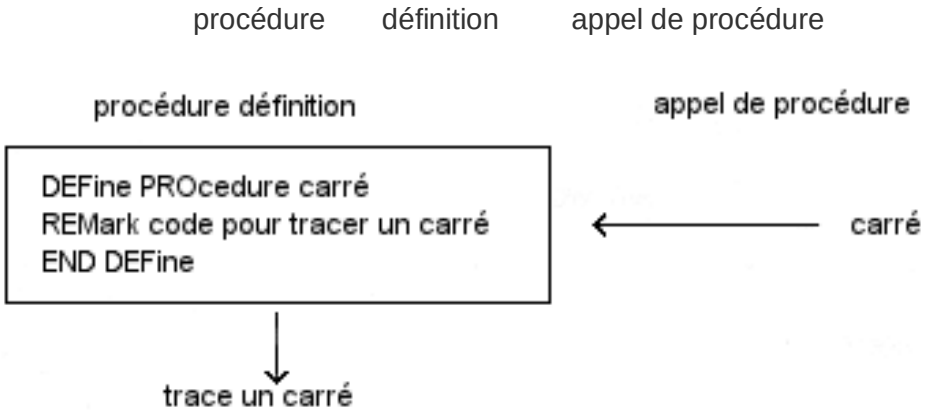
L'organisation des affaires les plus complexes de la vie courante procède de la même façon. C'est à la délégation de responsabilités que l'on reconnaît un bon gouvernement. Le Premier Ministre répartit le travail entre les ministres qui le divisent à leur tour jusqu'à ce que les tâches puissent être exécutées. Il peut y voir des complications dues aux services communs et aux interférences entre les différents niveaux, mais c'est tout de même cette structure hiérarchique qui domine.

Un bon programmeur travaillera aussi de cette manière, et un langage moderne, tel que le SuperBASIC qui emploie des procédures précisément nommées et bien définies sera beaucoup plus pratique que les anciennes versions qui n'ont pas ces possibilités.

L'idée est qu'un bloc séparé doit être écrit pour chaque tâche particulière. L'emplacement de ce bloc dans le programme n'a pas d'importance. Si on en utilise un, son emploi se fait ainsi :

- appel du bloc
- retour au point du programme situé juste après.

Ecrivons une procédure carré qui trace un carré. Voici le schéma :



Dans la pratique, les tâches séparées à l'intérieur d'un travail sont identifiées et nommées avant l'écriture de leur code de définition. Il n'y a que le nom qui soit nécessaire dans un appel de procédure, ainsi les lignes principales du programme peuvent être écrites avant que toutes les tâches soient écrites.

Mais si on le désire, on peut commencer par écrire les tâches et les tester. Si elles tournent, vous pouvez alors en oublier les détails et vous souvenir seulement de leur nom et de ce que fait la procédure.

On pourrait aisément écrire l'exemple suivant sans procédures, mais il est intéressant de voir comment on les utilise, même dans un cas relativement simple. Presque toutes les tâches peuvent être divisées de la même manière, aussi, n'avez-vous jamais à vous soucier de plus de cinq à trente lignes à la fois. Si vous savez correctement écrire trente lignes de programme, et créer des procédures, alors vous êtes parfaitement capable d'écrire trois cents lignes de programme. Il s'agit

de fabriquer des phrases toutes faites pour hommes politiques ou autres, qui veulent donner l'impression d'une grande aisance technologique sans pour autant connaître leur sujet. Chargez les mots suivants dans trois tableaux et écrivez un programme qui sort dix phrases toutes faites, aléatoires.

adjec1$	adjec2$	noun$
Full	fifth-generation	systems
Systematic	knowledge-based	machines
Intelligent	compatible	computers
Controlled	cybernetic	feedback
Automated	user-friendly	transputers
Synchronised	parallel	micro-chips
Functional	learning	capability
Optional	adaptable	programming
Positive	modular	packages
Balanced	structured	databases
Integrated	logic-oriented	spreadsheets
Coordinated	file-oriented	word-processors
Sophisticated	Standardised	objectives

Analyse :

Nous voulons écrire un programme qui sort des phrases toutes faites. Les étapes du programme sont :

1. Charger les mots dans trois tableaux chaînes.
2. Choisir trois nombres aléatoires qui seront les indices du tableau de variables.
3. Ecrire la phrase.
4. Refaire les étapes 2 et 3 dix fois.

Variables :

Nous déclarons trois tableaux dont les premiers contiendront les adjectifs ou les mots utilisés comme adjectifs. Le troisième tableau contiendra les noms. Il y a 13 mots dans chaque tableau, et le plus long a 16 caractères y compris le trait d'union.

Tableau	Contenu
adjec1$(13,12)	premiers adjectifs
adjec2$(13,16)	seconds adjectifs
noun$(13,15)	nous

Procédures :

Nous utilisons trois procédures :

- *store_data* charge les trois suites de 13 mots
- *get_random* fournit trois nombres aléatoires compris entre 1 et 13.
- *make_phrase* écrit une phrase.

Programme principal :

Le programme principal est très simple parce que le plus gros travail est fait par les procédures.

Déclarer (**DIM**) les tableaux
Store_data
Pour dix phrases
get_random
make_phrase
FIN

```
100 REMark ************
110 REMark * Buzzword *
120 REMark ************
130 DIM adjec1$(13,12), adjec2$(13,16), noun$(13,15)
140 store_data
150 FOR phrase = 1 TO 10
160 get_random
170 make_phrase
180 END FOR phrase
190 REMark *************************
200 REMark * Procedure Definitions *
210 REMark *************************
220 DEFine PROCedure store_data
230 REMark *** procedure to store the buzzword data ***
240 RESTORE 420
250 FOR item = 1 TO 13
260READ adjec1$(item), adjec2$(item), noun$(item)
270 END FOR item
280 END DEFine
290 DEFine PROCedure get_random
300 REMark *** procedure to select the phrase ***
310 LET ad1 = RND(1 TO 13)
320 LET ad2 = RND(1 TO 13)
330 LET n = RND(1 TO 13)
340 END DEFine
350 DEFine PROCedure make_phrase
360 REMark *** procedure to print out the phrase ***
370 PRINT ! adjec!$(ad1) ! adjec2$(ad2) ! noun$(n)
380 END DEFine
390 REMark ****************
400 REMark * Program Data *
410 REMark ****************
420 DATA "Full", "fifth-generation", "systems"
430 DATA "Systematic", "knowledge-based", "machines"
440 DATA 'Intelligent","compatible", "computers"
450 DATA "Controlled", "cybernetic", "feedback"
460 DATA "Automated", "user-friendly", "transputers"
470 DATA "Synchronised", "parallel", "micro-chips"
480 DATA "Functional", "Learning", "capability'
490 DATA "Optional", "adaptable", "programming"
500 DATA "Positive" , "modular" , "packages"
510 DATA "Balanced" , "structured", "databases"
520 DATA "Integrated", "logic-oriented", "spreadsheets"
530 DATA "Coordinated", "file-oriented", "word-processors"
540 DATA "Sophisticated", "standardised", "objectives"
```

Résultat :

```
Automated fifth-generation capability
Functional learning packages
Full parallel objectives
Positive user-friendly spreadsheets
Intelligent file-oriented capability
Synchronised cybernetic transputers
Functional logic-oriented micro-chips
Positive parallel feedback
Balanced learning databases
Controlled cybernetic objectives
```

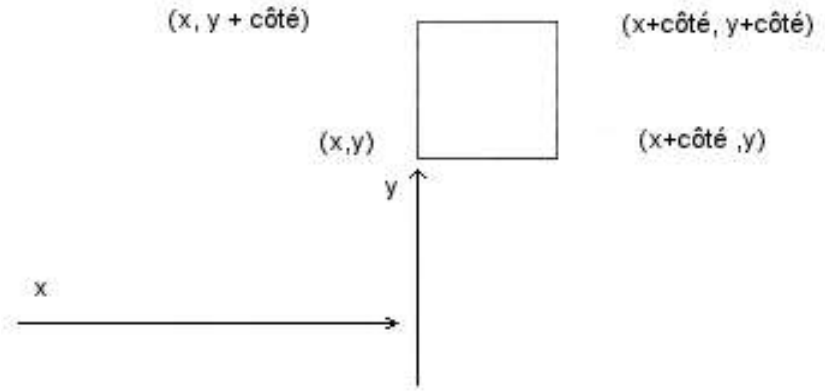
ENTRER DES DONNEES DANS LES PROCEDURES

Supposons que nous désirions dessiner des carrés de tailles, couleurs et emplacements différents.

Si nous définissons une procédure carré qui fait cela, il faudra lui fournir quatre informations :

- longueur d'un côté
- couleur (code couleur)
- position (abscisse et ordonnée du point de départ)

En effet la position du carré est déterminée par deux valeurs, l'abscisse x et l'ordonnée y du coin gauche du carré comme on l'indique ci-dessous :



Les couleurs du carré sont celles des côtés.

```
200 DEFine PROCedure carre(cote,x,y)
210   LINE x,x TO x+cote,y
220   LINE TO x+cote,y+cote
230   LINE TO x,y+cote TO x,y
240 END DEFine
```

Avant de lancer cette procédure, il faut fournir les valeurs de x, de y et de côté. C'est au moment de l'appel de la procédure qu'il faut les fournir. Vous pourriez, par exemple, avoir le programme principal suivant pour obtenir un carré vert de côté 20.

```
100 PAPER 7:CLS
110 INK 4
120 carre 20,50,50
```

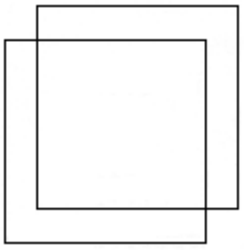
Les nombres 20, 50, 50 sont appelés paramètres et ils sont transférés dans les variables nommées dans la procédure.

carre 20,50,50

DEFine PROCedure (côté, x, y)

Les nombres 20, 50, 50 sont appelés des paramètres locaux. Ici, ce sont des nombres, mais ce pourrait aussi être des expressions. Les variables côté, x, y, sont appelés des paramètres formels. Ils doivent être des variables car ils reçoivent des valeurs.

Un programme principal plus intéressant utilise la même procédure pour créer une figure aléatoire de paires de carrés de couleurs. Chaque paire de carrés est obtenue en diminuant l'abscisse et l'ordonnée du second de un cinquième du côté du carré, comme ci-dessous :



```
100 REMark Figure de carrés
110 PAPER 7 : CLS
120 FOR paire = 1 TO 20
130   INK RND(5)
140   LET côté = RND(10 TO 20)
150   LET x = RND(50) : y = RND(70)
160   carré côté,x,y
170   LET x=x+côté/5 : y = y+côté/5
180   carré côté,x,y
190 END FOR paire
```

Les avantages des procédures sont :

1. Vous pouvez utiliser la même procédure plus d'une fois dans le même programme, ou dans d'autres.

2. Vous pouvez diviser une tâche en sous-tâches, et écrire des procédures pour chaque sous-tâche. Ceci vous aide à faire l'analyse de votre problème.

3. Chaque procédure peut être testée séparément. Ceci vous aide dans les tests et la suppression des erreurs.

4. Les procédures commencent et se terminent par des noms expressifs, ce qui vous aide à créer des programmes très lisibles.

Si vous utilisez des noms de procédures bien appropriés avec de bons paramètres, vous constaterez que votre capacité à résoudre les problèmes et à programmer est largement améliorée.

TEST SUR LE CHAPITRE 7.

Vous pouvez gagner un maximum de 14 points dans le test suivant. Calculer votre score à l'aide des réponses de la page suivante :

1. Comment aborde-t-on le problème de complexité de haut niveau dans la vie courante ?

2. Comment cette méthode peut-elle être appliquée en programmation ?

3. Quels sont les deux traits caractéristiques de la définition d'une simple procédure ? (2 pts)

4. Quelles sont les deux principales conséquences d'un appel de procédure ? (2 pts)

5. Quel est l'avantage de l'emploi de noms de procédures dans un programme principal avant l'écriture de la définition de ces procédures ?

6. Quel est l'avantage d'écrire une définition de procédure avant d'utiliser son nom dans un programme principal ?

7. Comment l'emploi de procédures peut-il aider un programmeur capable d'écrire des programmes d'une trentaine de lignes à en écrire de beaucoup plus longs ?

8. Certains programmes utilisent plus de mémoire en utilisant des procédures, mais dans quel cas est-ce que l'emploi des procédures économise de l'emplacement mémoire ?

9. Nommez deux moyens de passer des informations du programme principal dans une procédure.(2 pts)

10. Qu'est-ce qu'un paramètre local ?

11. Qu'est-ce qu'un paramètre formel ?

REPONSES AU TEST DU CHAPITRE 7.

1. Généralement, on divise les travaux importants et complexes en plus petites tâches, jusqu'à ce qu'elles soient suffisamment petites pour être exécutées.

2. Ce principe peut être appliqué en programmation en divisant le travail complet et en écrivant une procédure pour chaque tâche.

3. Une simple procédure est :
   - un bloc séparé
   - un nom bien approprié. (2 pts)

4. Par un appel de procédure on a :
   - activation de la procédure,
   - retour au point qui suit le point d'appel. (2 pts)

5. Les noms de procédures peuvent être utilisés dans un programme principal avant que les procédures soient écrites. Ceci vous permet de réfléchir à votre travail dans sa totalité, et vous donne une vue d'ensemble en vous évitant de vous soucier des détails.

6. Si vous écrivez une définition de procédure avant d'utiliser son nom, vous pouvez la tester et si elle fonctionne correctement, vous en oubliez alors les détails. Il vous suffit de retenir son nom, et, en gros, ce qu'elle fait.

7. Un programmeur capable d'écrire une trentaine de lignes de programme peut découper un problème complexe en plusieurs procédures, d'une trentaine de lignes maximum. De cette manière il n'a à se soucier que d'une petite partie du programme à la fois.

8. L'emploi d'une procédure peut économiser de la place en mémoire s'il est nécessaire de l'appeler plus d'une fois à partir de différents endroits du programme. La définition d'une procédure est écrite une seule fois, mais elle peut être appelée aussi souvent que nécessaire.

9. Un programme principal peut placer des informations dans des cases mémoires à l'aide des instructions **LET** ou **READ**. Ces cases mémoires sont accessibles à la procédure. Ainsi, la procédure peut utiliser les informations qui sont déjà dans le programme principal.

   Une deuxième méthode est d'utiliser des paramètres dans l'appel de procédure. Ces valeurs sont transférées aux variables de la définition de procédure, qui les utilise si nécessaire. (2 pts)

10. Un paramètre local est la valeur temporaire passée par l'appel de procédure dans le programme principal.

11. Un paramètre formel est une variable de la définition d'une procédure qui reçoit la valeur transférée dans la procédure par le programme principal.

Calculez votre score :

Ceci est un test de recherche. Il vous faut une plus grande expérience de l'emploi des procédures pour pouvoir les apprécier. Mais c'est un moyen très puissant qui, bien compris, inspire des idées extrêmement constructives. Il est très valable, même s'il demande un gros effort.

12 à 14 : Excellent - Continuez la lecture en toute confiance.

10 ou 11 : Très bien. Vérifiez seulement certains points.

8 ou 9 : Bien, mais relisez quelques passages du chapitre 7.

6 ou 7 : Convenable, mais relisez quelques passages du chapitre 7.Travaillez attentivement les programmes donnés en exemple, et changez les valeurs des variables. Puis refaites le test.

Moins de 6 : Relire le chapitre 7. Reprenez lentement en travaillant les programmes. Ces notions peuvent vous paraître difficiles, mais elles justifient l'effort. Quand vous serez prêt, refaites le test.

EXERCICES SUR LE CHAPITRE 7.

1. Ecrire un programme qui donne en sortie un des quatre arrangements possibles avec : 'Cœur', 'Trèfle', 'Carreau', 'Pique'. Appelez la procédure cinq fois pour obtenir cinq arrangements aléatoires.

2. Refaire l'exercice 1 en utilisant un nombre compris entre 1 et 4 comme paramètre pour déterminer le mot en sortie. Lorsque vous aurez fait cela, écrivez le programme sans paramètre.

3. Ecrire un programme qui sortira la valeur d'une carte soit sous forme d'un nombre compris entre 2 et 10, soit un des mots 'As','Valet', 'Dame',' Roi'.

4. Ecrire un programme qui appelle cette procédure cinq fois, donnant ainsi cinq valeurs aléatoires.

5. Réécrivez le programme de l'exercice 3 en utilisant un nombre compris entre 1 et 13 comme paramètres à faire passer dans la procédure. Essayez ensuite d'écrire le programme sans paramètre.

6. Ecrivez le programme le plus élégant possible, utilisant des procédures pour sortir quatre groupes de cinq cartes chacune. Ne vous inquiétez pas des cartes en double. L'élégance du programme viendra à la fois de la lisibilité, de la brièveté, et de l'efficacité. Chacun, en chaque circonstance, accordera à ces trois facteurs une importance qui, quelquefois, va à l'encontre de celle des autres.

CHAPITRE 8

DU BASIC AU SUPERBASIC

INTRODUCTION

Si vous connaissez bien l'une des dernières versions du BASIC, vous pouvez être tenté de sauter les sept premiers chapitres. Le présent chapitre vous servira alors de lien entre vos connaissances actuelles et les chapitres suivants. Si vous faites ainsi, et si vous rencontrez quelques difficultés, il peut être utile de faire un retour en arrière de quelques chapitres.

Si vous avez travaillé tout au long des précédents chapitres, la lecture de celui-ci vous paraîtra facile. Vous constaterez que, autant qu'une introduction à quelques idées nouvelles, il vous donne une notion intéressante de l'évolution du BASIC. En dehors de la programmation structurée, Super-BASIC est un net progrès en ce qui concerne une bonne présentation de l'écran, l'édition, les capacités graphiques et de traitement. En bref, c'est une combinaison de commodité et de puissance de traitement qui n'avait encore jamais existé.

Ainsi, lorsque vous faites la transition de BASIC vers SuperBASIC, vous allez non seulement vers un langage plus puissant et plus pratique, mais vous allez aussi vers un environnement informatique plus évolué.

Nous allons maintenant présenter les principales propriétés de SuperBASIC et les propriétés qui le distinguent des autres BASIC.

COMPARAISON DE TERMES ALPHABETIQUES

Les comparaisons usuelles sont possibles sur les chaînes de caractères.

```
LET pet1$ = "CHAT"
LET pet2$ = "CHIEN"
IF pet1$ < pet2$ THEN PRINT "Miaou"
```

Vous avez en sortie "Miaou" car le symbole < signifie « vient avant dans l'alphabet ». De même 'ERD98L' vient avant 'ERD746L'.

```
100 INPUT item1$, item2$
110 IF item1$ < item2$ THEN PRINT item1$
120 IF item1$ = item2$ THEN PRINT "égal"
130 IF item1$ > item2$ THEN PRINT item2$
```

ENTREE		SORTIE
chat	chien	chien
chat	CHIEN	chat
ERD98L	ERD746L	ERD98L
ABC	abc	ABC

Le guide de Référence vous fournira de nombreux détails sur les comparaisons de chaînes en SuperBASIC.

VARIABLES ET IDENTIFIANTS

La plupart des BASICs ont des variables numériques et des variables chaînes. Comme dans les autres BASICs, c'est le signe $ qui distingue le nom d'une variable chaîne. Il est toujours placé à la fin du nom :

Variables numériques :	compte	string:	*mot$*
	somme		*high_st$*
	total		*jour_de_la_semaine$*

Les règles des identifiants en SuperBASIC sont exposées dans le Concept Référence Guide. La longueur maximale d'un identifiant est 255 caractères. Votre choix d'identifiants est personnel. Quelquefois, les plus longs sont plus pratiques parce qu'ils transmettent au lecteur ce que fait un programme. Mais ils doivent être tapés, et il est fastidieux de taper des mots trop longs. Les mots les plus courts seront préférables, à. condition qu'ils expriment le sens de ce qu'ils représentent. Mais les mots très courts, ainsi que les lettres isolées devront être utilisés avec modération. Les noms de variables tels que X, Z, P3, Q2 introduisent un niveau d'abstraction que certains trouvent peu pratique.

VARIABLES ENTIERES

SuperBASIC distingue les variables entières, et les distingue avec le signe %. Ainsi on peut avoir :

- compte%
- nombre%
- prix_approximatif%

Nous avons maintenant deux sortes de variables numériques. Nous appelons les autres, qui peuvent avoir une virgule, des nombres à virgule flottante. Ainsi peut-on écrire :

```
LET prix = 9
LET cout = 7.31
LET compte% = 13
```

Mais si vous écrivez :

```
LET compte% = 5.43
```

la valeur de compte% sera 5.

Autre exemple :

```
LET compte% = 5.73
```

la valeur de compte% sera 6. Vous pouvez constater que SuperBASIC arrondit au nombre entier le plus proche.

RESTRICTION

SuperBAS1C essaie toujours d'être pratique et intelligent et n'envoie pas de message d'erreur comme le font d'autres BASIC dans les cas de ce genre : une variable chaîne, appelée mark$ a la valeur '64'. Ainsi :

```
LET score = mark$
```

vous donnera en sortie 64. C'est seulement si la chaîne ne peut pas être convertie que vous aurez un message d'erreur.

VARIABLES LOGIQUES ET SIMPLES PROCEDURES

Il existe un autre type de variable en SuperBASIC ou plus exactement SuperBASIC en donne l'impression. Considérons l'instruction en SuperBASIC :

```
IF vent THEN cerf_volant
```

Dans les autres BASICs on écrirait :

```
IF v=1 THEN GOSUB 300
```

Dans ce cas v=1 est une condition de l'expression logique qui ne peut être que vraie ou fausse. Si elle est vraie, alors une sous-routine l'exécutera à partir de la ligne 300. Cette sous-routine peut traiter le problème d'un cerf-volant, mais la ligne ci-dessus ne l'exprime pas. Un programmeur attentif rendrait l'instruction plus claire en écrivant :

```
IF v=1 THEN GOSUB 300 : REM cerf volant
```

dans un souci de lisibilité. Mais l'instruction SuperBASIC est parfaitement lisible en elle-même. L'identifiant vent est interprété comme vrai ou faux même s'il s'agit d'une variable à virgule flottante. Toute valeur non nulle sera considérée comme vraie. Zéro sera considéré comme faux. Ainsi, le simple mot vent joue le même rôle qu'une condition dans une expression logique.

L'autre mot cerf_ volant est une procédure. Il procède comme un GOSUB, mais en bien meilleur.

Le programme suivant vous donnera une idée de l'emploi de variables logiques et du type le plus simple de procédure.

```
100 INPUT vent
110 IF vent THEN cerf_volant
120 IF NOT vent THEN ranger_hangar
130 DEFine PROCedure cerf_volant
140 PRINT "Regarder dans le ciel."
150 END DEFine
160 DEFine PROCedure ranger_hangar
170 PRINT "Jeter les ordures."
180 END DEFine
```

ENTREE	SORTIE
0	jeter les ordures
1	regarder le ciel
2	regarder le ciel
-2	regarder le ciel

Vous le voyez, il n'y a que zéro qui signifie faux. Si les procédures ne contiennent qu'une instruction chacune, c'est pour situer l'exemple dans un contexte simple. On parlera plus longuement des procédures dans la suite de ce chapitre.

L'INSTRUCTION LET

En SuperBASIC, **LET** est facultatif, mais nous l'utilisons dans ce manuel pour éviter tout risque de confusion entre les deux emplois du signe =

```
LET compte = 3
```

et

```
IF compte = 3 THEN EXIT
```

Ces instructions sont différentes et le terme **LET** aide à les distinguer. Toutefois, s'il y a deux ou plusieurs instructions LET exécutant un travail simple tel que l'initialisation de variables, on peut faire une exception

Par exemple :

```
100 LET premier = 0
110 LET second = 0
120 LET troisième = 0
```

peut s'écrire :

```
100 LET premier = 0 : second = 0 : troisième = 0
```

sans perte dans la clarté du style. Cette méthode peut aussi s'utiliser avec d'autres instructions, afin d'obtenir une forme abrégée. Les deux points : sont un terminateur d'instructions et peuvent être utilisés avec d'autres instructions que **LET**.

L'ÉCRAN BASIC MODES ET PIXELS

Dans un prochain chapitre nous expliquerons comment tracer des figures géométriques, telles que des cercles, mais ici nous présentons les propriétés des pixels. Il existe deux modes qui peuvent être employés dans les deux cas suivants :

Basse résolution	
Mode 8 couleurs	**MODE 256**
512 pixels en abscisse	**MODE 8**
256 pixels en ordonnée	

Haute résolution	
Mode 4 couleurs	**MODE 512**
512 pixels en abscisse	**MODE 4**
256 pixels en ordonnée	

Dans les deux modes on peut adresser les pixels dans les intervalles :

- abscisse 0 à 511
- ordonnée 0 à 255

Puisque le mode 0 a seulement la moitié du nombre de pixels en abscisse de l'écran du mode 1, en mode 0, les pixels sont deux fois plus larges qu'en mode 1, ainsi en mode 0, chaque pixel peut être spécifié par deux coordonnées :

- 0 ou 1 2 ou 3 510 ou 511

Cela signifie aussi que vous utiliserez les mêmes nombres pour adresser des pixels, sans tenir compte du mode. Pensez toujours : de 0 à 511 en abscisse, de 0 à 255 en ordonnée.

Si vous disposez d'un téléviseur, alors les pixels ne seront pas tous visibles.

COULEURS

Les couleurs disponibles sont :

MODE 256	Code	MODE 512
Noir	0	Noir
Bleu	1	
Rouge	2	Rouge
Magenta	3	
Vert	4	Vert
Cyan	5	
Jaune	6	Blanc
Blanc	7	

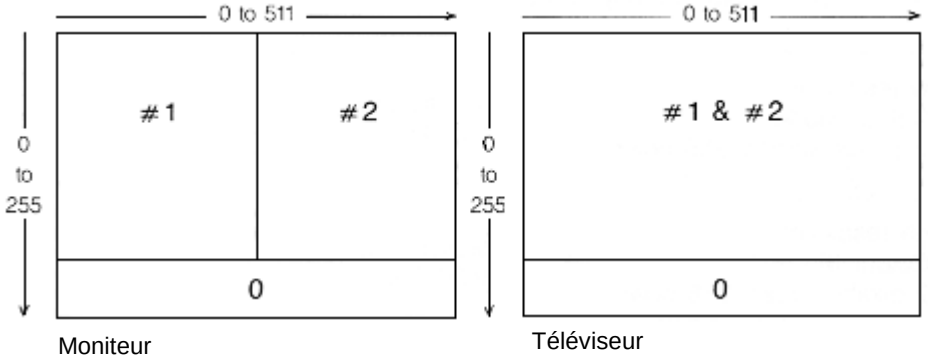
Dans le mode haute résolution, on peut sélectionner chaque couleur avec l'un des deux codes. Vous verrez plus tard comment il est possible d'obtenir une quantité étonnante de couleurs et des effets de fond si vous disposez d'un moniteur de bonne qualité.

Les principaux mots-clés de la présentation de l'écran sont les suivants :

INK *couleur*	couleur des traits et caractères
BORDER *largeur, couleur*	trace une bordure autour de l'écran ou d'une fenêtre.
PAPER *couleur*	couleur de fond
BLOCK *abscisse, ordonnée, largeur, hauteur, couleur*	colore un rectangle dont le sommet gauche est en position (abscisse, ordonnée)

ORGANISATION DE L'ECRAN QL

Quand vous mettez en marche votre QL, la surface de l'écran est divisée en trois parties appelées fenêtres, comme sur la figure ci-dessous :



Les fenêtres sont identifiéespar #0, #1, #2, ainsi vous pouvez intervenir dans les différentes fenêtres:

Par exemple :

```
CLS
```

effacera la fenêtre #1, aussi si vous désirez effacer la fenêtre de gauche

vous taperez :

```
CLS #2
```

Si vous désirez une couleur différente du fond, tapez pour le vert :

```
PAPER 4 : CLS
```

ou

```
PAPER #2,4 : CLS #2
```

pour effacer la fenêtre #2 et la mettre au fond vert.

Ces numéros #0, #1, #2 sont appelés des numéros de canaux. Dans ce cas particulier, ils vous permettent de diriger certains paramètres dans la fenêtre de votre choix. Vous verrez par la suite que les numéros de canaux ont beaucoup d'autres applications, mais pour l'instant, notez bien que toutes les instructions suivantes doivent avoir un numéro de canal. La troisième colonne vous indique le canal pris par défaut, c'est à dire celui qui est choisi par le système si vous n'en spécifiez aucun.

Notez que les fenêtres peuvent se chevaucher. Si vous disposez d'un écran TV, le système fait chevaucher de façon automatique les fenêtres #1 et #2, ce qui permet d'avoir plus de positions de caractères par ligne de disponibles pour les listings des programmes.

MOT-CLE	EFFET	PAR DEFAUT
AT	Position du caractère	#1
BLOCK	Colore une surface	#1
BORDER	Trace une bordure	#1
CLS	Nettoie l'écran	#1
CSIZE	Taille du caractère	#1
CURSOR	Position du curseur	#1
FLASH	Provoque des arrêts de clignotement	#1
INK	Couleur des caractères et des figures	#1
OVER	Permet l'écriture et les graphiques	#1
PAN	Déplace l'écran latéralement	#1
PAPER	Couleur du fond	#1
RECOL	Change la couleur	#1
SCROLL	Déplace l'écran verticalement	#1
STRIP	Couleur du fond pour l'impression	#1
UNDER	Souligne	#1
WINDOW	Modifie une fenêtre existante	#1
LIST	Affiche un programme	#2
DIR	Affiche un répertoire	#1
PRINT	Affiche des caractères	#1
INPUT	Prend les entrées clavier	#1

Les instructions et les commandes directes apparaissent dans la fenêtre #0.

Pour avoir plus de détails au sujet de la syntaxe de ces mots clés, consultez la partie correspondante du manuel.

RECTANGLES ET DROITES

Le programme ci-dessous construit un rectangle vert en mode 256 sur fond rouge, avec une bordure jaune d'un pixel de large. Le rectangle a son sommet gauche au centre de l'écran (256,128). Il a une largeur de 140 unités (70 pixels) et sa hauteur est de 100 unités (100 pixels).

```
100 REMark Rectangle
110 MODE 256
120 BORDER 1,6
130 PAPER 2 : CLS
140 BLOCK 140,100,256,128,4
```

Faites bien attention au MODE 256 parce que la valeur des abscisses varie de 0 à 511, même s'il n'y a que 256 pixels. On ne peut donc pas dire que le rectangle construit par le programme ci-dessus a 140 pixels de large, aussi dit-on 140 unités.

ENTREES ET SORTIES

SuperBASIC utilise pour les entrées les instructions **LET**, **INPUT**, **READ**, **DATA**. L'instruction **PRINT** effectue les sorties ; elles sont souvent utilisées avec les séparateurs :

,	Sort d'une tabulation
;	Ne fait rien d'autre que séparer
\	Implique un saut à la ligne suivante
!	Espace intelligent. Il provoque normalement un espace, mais non en début de ligne. Si un paragraphe est trop long pour une ligne, génère une nouvelle ligne.
TO	Place une tabulation à la colonne spécifiée

BOUCLES

Vous devez vous familiariser aux deux types de boucles ci-dessous :

a) Simuler six coups d'un dé à six faces.

```
100 FOR coup = 1 TO 6
110   PRINT RND(1 TO 6)
120 NEXT coup
```

b) Simuler des coups de dés jusqu'à l'apparition d'un six.

```
100 dé = RND(1 TO 6)
110 PRINT dé
120 IF dé <> 6 THEN GOTO 10
```

Ces deux programmes sont corrects en SuperBASIC, mais nous vous recommandons plutôt les suivants.

Ils font exactement la même chose. Bien que le programme b) soit un peut plus compliqué, il y a de bonnes raisons de le préférer.

Programme a)

```
100 FOR coup = 1 TO 6
110 PRINT RND(1 TO 6)
120 END FOR coup
```

Programme b)

```
100 REPeat coups
110 dé = RND(1 TO 6)
120 PRINT dé
130 IF dé = 6 THEN EXIT coups
140 END REPeat coups
```

Il est logique de disposer d'une structure de boucle qui se termine par une condition (boucles **REPeat**), de la même façon que pour celles qui sont contrôlées par un compteur.

La structure fondamentale de **REPeat** est :

REPeat identifiant
  instructions
END REPeat identifiant

On peut placer une instruction **EXIT** n'importe où dans la structure, mais elle doit être suivie d'un identifiant qui indique à SuperBASIC de quelle boucle il faut sortir ; par exemple :

EXIT coups

donnera le contrôle à l'instruction qui suit :

END REPeat coups

On pourrait penser que ce simple problème est illustré ici de façon exagérée. Mais la structure **REPeat** est très puissante. Elle vous aidera dans bien des cas. Si vous connaissez d'autres langages, vous remarquerez qu'elle remplace à la fois les structures **REPeat** et **WHILE** et permet ainsi de résoudre toutes les situations.

La boucle **REPeat** de SuperBASIC porte un nom, ce qui permet d'en sortir sans erreur possible. La boucle **FOR**, comme toutes les structures de SuperBASIC, se termine par **END** et on décrira par la suite les raisons du choix de son nom.

Vous verrez aussi plus tard comment on peut utiliser ces structures de boucles dans des situations simples ou complexes pour faire exactement ce que vous désirez. Elles vous deviendront familières si vous êtes un utilisateur expérimenté du BASIC.

L'incrément de la variable de contrôle d'une boucle **FOR** est normalement 1 mais vous pouvez lui donner d'autres valeurs en utilisant le mot-clé **STEP**, comme le montre l'exemple suivant :

Exemple 1 :

```
100 FOR température = 2 TO 10 STEP 2
110    PRINT ! température !
120 END FOR température
```

Vous aurez en sortie 2, 4, 6, 8, 10

Exemple 2 :

```
100 FOR recul = 9 TO 1 STEP -1
110    PRINT ! recul !
120 END FOR recul
```

Vous aurez en sortie : 9 8 7 6 5 4 3 2 1

La seconde particularité et que les boucles peuvent être emboitées. Par exemple, le programme suivant donne en sortie quatre rangées de dix croix.

```
100 REMark croix
110 FOR rangée = 1 TO 4
120 PRINT 'Rangée numéro' ! rangée
130  FOR croix = 1 TO 10
140  PRINT ! 'X' !
150 END FOR croix
160 PRINT
170 PRINT \ 'Fin de la rangée numéro' ! rangée
180 END FOR rangée
```

Vous aurez en sortie :

```
Rangée numéro 1
X X X X X X X X X X
Fin de la rangée 1
Rangée numéro 2
X X X X X X X X X X
Fin de la rangée 2
Rangée numéro 3
X X X X X X X X X X
Fin de la rangée 3
Rangée numéro 4
X X X X X X X X X X
Fin de la rangée 3
```

Un gros avantage du SuperBASIC est qu'il a une structure pour tous les types de boucles, pas seulement pour la boucle **FOR** et ces boucles peuvent être emboitées l'une dans l'autre selon les besoins de la tâche. Il est possible d'emboiter une boucle **REPeat** dans une boucle **FOR**. Le programme ci-dessous fournit les scores de deux dés dans chaque rangée, jusqu'à ce que le score soit 7, à la place des croix.

```
100 REMark Rangée de dé
110 FOR rangée = 1 TO 4
120 PRINT 'Rangée numéro '! rangée
130 REPeat coups
140   LET dé1 = RND(1 TO 6)
150   LET dé2 = RND(1 TO 6)
160   LET score = dé1 + dé2
170   PRINT ! score !
180   IF score = 7 THEN EXIT coups
190 END REPeat coups
200 PRINT \'Fin de rangée numéro '! rangée
210 END FOR rangée
```

Exemple de sortie

```
Rangée numéro 1
8 11 6 3 7
Fin de rangée numéro 1
Rangée numéro 2
4 6 2 9 4 5 12 7
Fin de rangée numéro 2
Rangée numéro 3
7
Fin de rangée numéro 3
Rangée numéro 4
6 2 4 9 9 7
Fin de rangée numéro 4
```

La troisième particularité des boucles de SuperBASIC est leur flexibilité dans l'écriture des valeurs de la boucle **FOR**. Le programme suivant illustre ceci en affichant tous les nombres divisibles de 1 à 20 (un nombre divisible est un nombre qui peut être divisé par un autre nombre que 1 et lui-même).

```
100 REMark Nombres divisibles
110 FOR nomb = 4,6,8, TO 10,12,14 TO 16,18, 20
120   PRINT ! nomb !
130 END FOR nomb
```

STRUCTURE DE DECISION

Vous avez sans doute remarqué ce type simple de décision :

```
IF dé = 6 THEN EXIT coups
```

Cette forme est disponible dans la plupart des BASICs, mais SuperBASIC vous offre des extensions de cette structure dont l'une, toute nouvelle, permet de gérer des situations qui contiennent plus de deux alternatives.

Ci-dessous vous avez deux exemples de la forme longue IF... THEN. Ces exemples s'expliquent par eux-mêmes.

Exemple 1 :

```
100 REMark Forme longue IF. ..END IF
110 LET soleil = RND(0 TO 1)
120 IF soleil THEN
130 PRINT 'Porter des lunettes de soleil'
140 PRINT 'Aller se promener'
150 END IF
```

Exemple 2 :

```
100 REMark Forme longue IF...ELSE...END IF
110 LET soleil = RND(0 TO 1)
120 IF soleil THEN
130 PRINT 'Porter des lunettes de soleil'
140 PRINT 'Aller se promener'
150 ELSE
160 PRINT 'Mettre un manteau'
170 PRINT 'Aller au cinéma'
180 END IF
```

Le séparateur THEN est facultatif dans la forme longue, et il peut être remplacé par : dans la forme courte. Les structures de décision en forme longue ont le même statut que les boucles. Vous pouvez les indiquer, ou placer une autre structure à l'intérieur. Si une condition contient seulement une variable, alors la valeur zéro sera prise comme fausse et toute autre valeur comme vraie.

SOUS-ROUTINES ET PROCEDURES

La plupart des BASICs emploient l'instruction **GOSUB** pour rappeler une suite d'instructions appelée sous-routine. Mais l'instruction **GOSUB** n'est pas satisfaisante dans un certain nombre de cas et SuperBASIC met à votre disposition les procédures aux noms bien appropriés et munies de quelques propriétés bien pratiques.

Considérez les deux programmes suivants qui tracent un carré vert de 50 pixel de côté au point d'abscisse 20 et d'ordonnée 10, sur fond rouge.

a) Avec **GOSUB**

```
100 LET couleur = 4 : fond = 2
110 LET abscisse = 20
120 LET ordonnée = 10
130 LET côté = 50
140 GOSUB 170
150 PRINT 'FIN'
160 STOP
170 REMark Routine pour dessiner un carré
180 PAPER fond : CLS
190 BLOCK côté, côté, abscisse, ordonnée, couleur
200 RETurn
```

b) Utilisons une procédure avec paramètres

```
100 carré 4, 50, 20, 10, 2
110 PRINT 'FIN'
120 DEFine PROCedure carré(couleur,côté,abscisse,ordonnée,fond)
130 PAPER fond : CLS
140 BLOCK côté, côté, abscisse, ordonnée, couleur
150 END DEFine
```

Dans le premier programme, les valeurs de couleur, abscisse, ordonnée, côté sont données par des instructions **LET**, avant que l'instruction **GOSUB** n'active les lignes 180 et 190. Le contrôle est ensuite renvoyé par l'instruction **RETURN**.

Dans le second programme, les valeurs sont données dans la première ligne comme paramètres de la procédure appelée « carré » qui tout à la fois active la procédure et produit les valeurs dont elle a besoin.

Dans sa plus simple forme, une procédure n'a pas de paramètre. Elle englobe simplement un certain nombre de lignes d'instructions, mais même dans cette plus simple forme, la procédure a l'avantage sur l'instruction GOSUB d'être nommée de façon bien appropriée et isolée dans un seul paragraphe.

La puissance des procédures, et leur propriété de simplification sont d'autant plus évidentes que les programmes sont plus longs. Le rôle des procédures lorsque les programmes deviennent plus longs est moins de rendre la programmation plus facile que de l'empêcher de devenir plus difficile avec l'augmentation de la taille du programme. Tout en restant dans un contexte très simple, les exemples ci-dessus illustrent bien cette idée.

EXEMPLES

Les exemples suivants indiquent le type de vocabulaire et de syntaxe qui ont été employés dans ce chapitre et les précédents et qui formeront la base de la seconde partie de ce manuel.

On donne les lettres d'un palindrome (mot pouvant être lu indifféremment de gauche à droite ou inversement : Laval, par exemple) comme de simples caractères alphabétiques par des instructions **DATA**. Le dernier caractère est un astérisque et vous ne connaissez pas le nombre de lettres du palindrome. On vous demande de lire les lettres par un tableau et de les écrire à l'envers. Quelques palindromes n'ont de sens que si les espaces et la ponctuation sont ignorés, comme dans « Elu par crapule ».

```
100 REMark Palindromes
110 DIM texte$(30): RESTORE
120 LET Pos = 30
140 REPeat lecture
150   READ CARACTERE$ : PRINT CARACTERE$
160   IF CARACTERE$ = '*' THEN EXIT lecture
170   LET Pos = Pos-1
180   LET texte$(Pos) = CARACTERE$
190 END REPeat lecture
200 PRINT : PRINT texte$(Pos TO 30)
210 DATA 'L','A','M','E','D','E','S','U','N','S','J','A','M','A','I','S'
220 DATA 'N','U','S','E','D','E','M','A','L','*'
```

Le programme suivant prend en entrée des nombres compris entre 1 et 3999 et les convertit en leur équivalent en chiffres romains. Il ne respecte pas complétement l'écriture de ces chiffes puisqu'il donne IIII au lieu de IV.

```
100 REMark CHIFFRES ROMAINS
110 CLS : INPUT 'Nombre : '!NOMBRE
120 RESTORE 210
130 FOR type = 1 TO 7
140   READ lettre$, valeur
150   REPeat ERITURE
160     IF NOMBRE < valeur : EXIT ECRITURE
170     PRINT lettre$;
180     LET NOMBRE = NOMBRE - valeur
190   END REPeat output
200 END FOR type
210 DATA 'M',1000,'D',500,'C',100,'L',50,'X',10,'V',5,'I',1
```

Vous avez intérêt à étudier les exemples ci-dessus très attentivement, en les modifiant au besoin, jusqu'à ce que vous soyez assuré de bien les avoir compris.

CONCLUSION

Le SuperBASIC est un langage structuré, aussi les éléments de programme se suivent-ils en séquence ou s'intercalent nettement l'un dans l'autre. Toutes les structures sont nommées et peuvent ainsi être identifiées par le système. Il contient de nombreux moyens d'unification et de simplification et un grand nombre d'autres facilités.

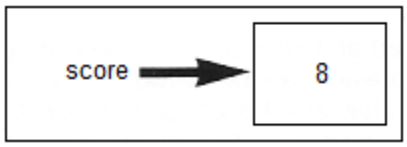
La plupart d'entre elles sont exposées et illustrées dans les chapitres suivants de ce manuel, qui seront, sans aucun doute plus faciles à lire que les parties "MOTS-CLÉS" et "CONCEPTS". Toutefois, s'ils sont plus faciles à lire, c'est parce qu'ils ne donneront pas tous les détails techniques et n'en développent pas en totalité tous les thèmes. Par conséquent, vous serez sans doute obligé, dans certains cas, de consulter quelques parties de la référence.

Néanmoins, quelques passages essentiels sont exposés dans les chapitres suivants. Peu de lecteurs en auront besoin, et vous pouvez en omettre quelques-uns, après une première lecture.

CHAPITRE 9

TYPES DE DONNEES VARIABLES ET IDENTIFIANTS

Vous avez sûrement remarqué qu'un programme (une suite d'instructions) utilise en général des données en entrée, et fournit plusieurs sortes de résultats en sortie. Vous avez aussi compris qu'il y a une organisation interne de chargement de ces données. Afin d'éviter d'entrer dans les détails techniques, nous vous proposons de vous représenter des cases mémoires dont vous aurez choisi les noms les plus expressifs. Par exemple, s'il est nécessaire de changer un nombre qui représente le score de deux jeux de dés fictifs, vous imaginez une case-mémoire nommée score qui pourrait contenir un nombre tel que 8.



Cette représentation est appelée une variable.

C'est le mot score qui la caractérise. Nous appelons ce mot un identifiant. C'est le mot que vous voyez, et il identifie le concept dont nous avons besoin, dans le cas présent, le résultat 8, d'un coup de deux dés. Comme l'identifiant est ce que nous voyons, c'est de lui que nous parlons, c'est lui que nous écrivons, c'est à lui que nous pensons. Nous pouvons écrire des instructions concernant score et ses valeurs à n'importe quel moment.

Il existe quatre types simples de données appelés nombres à virgule flottante, entiers, chaînes, et expressions logiques ; elles sont décrites ci-dessous. Nous parlons de types de données plutôt que de types de variables, comme 3.4 ou "chapeau bleu". Mais si vous connaissez les différents types de variables, vous connaîtrez aussi les différents types de données.

IDENTIFIANTS ET VARIABLES

1. Un identifiant SuperBASIC doit commencer par une lettre et ne peut contenir que :
   - des caractères majuscules ou minuscules
   - des nombres ou le caractère.

2. Un identifiant peut avoir jusqu'à 255 caractères, ainsi, sa longueur n'est pratiquement pas limitée.

3. Un identifiant ne doit pas utiliser l'un des mots-clés de SuperBASIC.

4. Le nom d'une variable entière est un identifiant se terminant par le caractère %.

5. Le nom d'une variable chaîne est un identifiant se terminant par le caractère $.

6. Les identifiants autres que les deux précédents ne peuvent se terminer par % ou $.

7. Un identifiant doit généralement être choisi en sorte qu'il signifie quelque chose pour le lecteur, mais pour SuperBASIC, il ne doit pas avoir d'autres sens que l'identification de la variable qu'il représente.

VARIABLES VIRGULES FLOTTANTES

Exemples de l'emploi de virgules flottantes :

```
100 LET jours = 24
110 LET ventes = 3649.84
120 LET ventes_par_jour = ventes/jours
130 PRINT ventes_par_jour
```

Les valeurs, d'une variable virgule flottante sont comprises dans l'intervalle :
$\pm 10^{-615}$ to $\pm 10^{+615}$ avec 8 chiffres significatifs.

Supposons que, dans le programme ci-dessus, les ventes soient exceptionnellement, seulement de 3 centimes. Changeons la ligne 110 :

```
20 LET ventes = 0.03
```

Le système traduira :

```
20 LET ventes = 3E-2
```

Pour comprendre ceci, partons de la valeur 3 et déplaçons le point décimal (la virgule en français) de deux rangs vers la gauche. Nous écrirons alors :

3E-2 qui équivaut à 0.03

Après exécution du programme, la moyenne journalière des ventes sera

1.25E-3 qui équivaut à 0,00125.

On dira que les nombres contenant un E sont exprimés sous forme de puissances de 10 :

(mantisse)E(exposant) = mantisse xl 0^n n étant l'exposant.

VARIABLES ENTIERES

Les variables entières sont des nombres entiers qui doivent être compris entre -32768 et +32767. Vous avez ci-dessous des noms de variables entières corrects, puisque terminés par %.

```
LET compte% = 10
LET six_comptes% = RND(10)
LET nombre_3% = 3
```

L'unique inconvénient des variables entières est le symbole % qui pourrait prêter à confusion : il n'a rien à voir avec la notion de pourcentage. C'est seulement un symbole bien pratique pour caractériser la variable entière.

FONCTIONS NUMERIQUES

L'utilisation d'une fonction ressemble à la confection d'une omelette. Vous utilisez un œuf qui est transformé selon certains processus en omelette. Par exemple, la fonction **INT** prend n'importe quel nombre en entrée, et donne en sortie sa partie entière. La valeur en entrée d'une fonction est appelée un paramètre ou un argument. **INT** est une fonction qui donne la partie entière d'une expression.

Vous pouvez écrire :

```
PRINT INT(5.6)
```

et vous aurez en sortie : 5. Nous dirons que 5.6 est le paramètre et que la fonction donne la valeur 5. Une fonction peut contenir plus d'un paramètre. Vous avez déjà rencontré :

```
RND(1 TO 6)
```

qui est une fonction à deux paramètres.

Mais les fonctions donnent toujours comme résultat une valeur unique qui est alors utilisée dans l'expression. Par exemple :

```
PRINT 2 * INT(5.6)
```

INT et **RND** sont des fonctions du système ; elles en font partie, mais vous verrez plus tard comment écrire vos propres fonctions.

L'exemple suivant montre une application courante de la fonction **INT**.

```
100 REMark Arrondi
110 INPUT décimal
120 PRINT INT(décimal + 0.5)
```

Dans cet exemple, vous entrez un nombre décimal et le résultat est arrondi à l'entrée le plus voisin. Ainsi 4.7 donnerait 5 alors que 4.3 donnerait 4.

Les fonctions trigonométriques seront indiquées dans une prochaine partie, mais d'autres fonctions numériques courantes sont énoncées dans la liste ci-dessous :

Fonction	Effet	Exemples	Valeurs produites
ABS	Valeur absolue	ABS(7) ABS(-4.3)	7 4.3
INT	Partie entière d'un nombre en virgule flottante	INT(2.4) INT(0.4) INT(-2.7)	2 0 -3
SQRT	Racine carrée	SQRT(2) SQRT(16) SQRT(2.6)	1.414214 4 1.612452

Il y a une méthode de calcul des racines carrées qui n'est pas difficile à comprendre. Pour calculer la racine carrée de 8 prenons une valeur approchée. Peu importe si elle est mauvaise. Prenons par exemple la moitié de 8 qui est 4.

Puisque 4 est plus grand que la racine carrée de 8, le quotient de 8 par 4 sera plus petit que la racine carrée. Inversement, si vous aviez pris 2 qui est plus petit que la racine carrée de 8, alors 8/2 donnerait un nombre plus grand que cette racine carrée.

Il en résulte qu'en prenant n'importe quel nombre et en divisant le nombre donnée par ce nombre, on obtient deux résultats, l'un plus petit, l'autre plus grand que la racine carrée. En faisant la moyenne de ces nombres nous nous rapprochons de la réponse correcte.

Nous répétons ce processus, jusqu'à ce que la différence entre le nombre et les approximations successives de la racine carrée soit nulle.

```
100 REMark Racine carrée
110 LET nombre = 8
120 LET approx = nombre/2
130 REPeat boucle
140   LET valeur_nouvelle = (approx + nombre/approx)/2
150   IF valeur_nouvelle == approx THE EXIT boucle
160   LET approx = valeur_nouvelle
170 END REPeat boucle
180 PRINT 'La recine carrée de ' ! nombre ! 'est' ! valeur_nouvelle
```

Vous aurez en sortie (par exemple pour le nombre 8) :

```
La racine carrée de 8 est 2.828427
```

Notez bien que la sortie conditionnelle de la boucle doit être située au milieu de la structure.

Le signe = = de la ligne 150 signifie " sensiblement égal à", soit égal à 10^{-7} près (ou 0,0000001).

OPERATIONS NUMERIQUES

SuperBASIC se charge des opérations mathématiques usuelles. Vous pourrez remarquer qu'elles procèdent comme les fonctions à deux opérandes. La convention est de placer un opérande de chaque côté du symbole. L'opération est exprimée soit par un signe comme + ,*, soit par un mot-clé tel que **DIV** ou **MOD**. Les opérations numériques doivent respecter un ordre de priorité. Par exemple le résultat de

```
PRINT 7 + 3*2
```

est 13 parce que la multiplication est prioritaire sur l'addition. Cependant :

```
PRINT (7+3)*2
```

donnera 20, parce que l'expression entre parenthèse doit être calculée avant toute autre opération. Comme vous le verrez plus tard, les expressions en SuperBASIC peuvent faire beaucoup de choses, et on ne peut pas tout voir à notre stade. (Consultez le Guide Référence des Concepts si vous le désirez). Mais les opérations que nous verrons maintenant devront respecter les priorités suivantes :

- d'abord élever à une puissance
- ensuite multiplier ou diviser (y compris avec **DIV** ou **MOD**) en dernier, additionner ou soustraire.

On peut aussi utiliser les symboles + et - avec un seul opérande, ils exprimeront simplement le signe du nombre. Dans ce cas, ces symboles sont prioritaires sur tout le reste, et il n'y a que les parenthèses qui peuvent avoir une plus grande priorité.

Finalement, si deux symboles ont la même priorité, on exécutera d'abord la plus à gauche ; dans

```
PRINT 7 - 2 + 5
```

la soustraction sera effectuée avant l'addition.

Opération	Symbole	Exemple	Résultat	Remarque
Addition	+	7+6.6	13.6	
Soustraction	-	7-6.6	0.4	
Multiplication	*	3*2.1	6.3	
		2.1*(-3)	-6.3	
Division	/	7/2	3.5	La division par zéro est impossible
		-17/5	-3.4	
Puissance	^	4^1.5	8	
Division entière	DIV	-8 DIV 2	-4	Valable seulement pour les nombres entiers
		7 DIV 2	3	La division par zéro est impossible
Modulo	MOD	13 MOD 5	3	
		21 MOD 7	0	
		-17 MOD 8	7	

Modulo donne le reste de la division. Toute tentative de division par zéro vous génère une erreur et termine l'exécution du programme.

EXPRESSIONS NUMERIQUES

A parler strictement, une expression numérique est une expression qui calcule un nombre, et il y a plus de possibilités qu'il nous en faut pour cet exposé. SuperBASIC vous permet de faire des choses complexes, mais aussi des toutes simples. Dans cette partie nous nous intéresserons aux règles mathématiques fondamentales.

En principe, les expressions numériques en SuperBASIC s'expriment comme en mathématiques, mais vous devez écrire l'expression entière sur une seule ligne :

$$\frac{5+3}{6-4}$$

deviendra en SuperBASIC (ou autres BASIC) :

(5 + 3)/(6 - 4)

Exemple 1 :

Dans l'équation du second degré : $ax^2 + bx + c = 0$

une solution en écriture mathématique est :

$$x = \frac{-b + \sqrt{b2 - 4ac}}{2a}$$

si nous prenons comme équation : $2x^2 - 3x + 1 = 0$

Le programme suivant trouve une solution :

```
100 READ a,b,c
110 PRINT 'La racine est' ! (-b+SQRT(b^2 - 4*a*c))/(2*a)
120 DATA 2,-3,1
```

Exemple 2 :

Pour les problèmes qui ont besoin de simuler la distribution d'un jeu de cartes, on peut faire correspondre des cartes aux nombres de 1 à 52 comme suit :

1 to 13
14 to 26
27 to 39
40 to 52
As,2 ..... Roi de cœur
As,2 ..... Roi de carreau
As,2 ..... Roi de trèfle
As,2 ..... Roi de pique

On peut identifier une carte de la manière suivante :

```
100 REM identification des cartes
110 INPUT 'Numéro de carte (compris entre 1 et 52)' ! carte
120 LET suite = (carte-1) DIV 13
130 LET valeur = card MOD 13
140 IF valeur = 0 THEN LET valeur = 13
150 IF valeur = 1 THEN PRINT "As de ";
160 IF valeur >= 2 AND valeur <= 10 THEN PRINT valeur! "de ";
170 IF valeur = 11 THEN PRINT "Vale de ";
180 IF valeur = 12 THEN PRINT "Dame de ";
190 IF valeur = 13 THEN PRINT "Roi de ";
200 IF suite = 0 THEN PRINT "coeur"
210 IF suite = 1 THEN PRINT "carreau"
220 IF suite = 2 THEN PRINT " trèfle "
230 IF suite = 3 THEN PRINT "pique"
```

La ligne 160 comporte une idée nouvelle. Le nombre contenu dans la variable « valeur » ne sera imprimé uniquement que dans le cas où les deux instructions logiques suivantes sont vraies :

valeur > =2 et valeur <= 10

Les cartes en dehors de cet intervalle sont soit des as, soit des valets, dames ou rois, et doivent être traitées autrement.

Remarquez aussi que **!** dans une instruction **PRINT** produit un espace et que **;** fait en sorte que le **PRINT** suivant s'écrira à la suite sur la même ligne.

Il y a deux groupes de fonctions mathématiques dont nous n'avons pas encore parlé. Ce sont les fonctions trigonométriques et logarithmiques. Mais tous les types de fonctions sont pleinement définis dans la partie référence.

VARIABLES LOGIQUES

A strictement parler, SuperBASIC n'a pas de variables logiques, mais il vous autorise à utiliser n'importe quelle variable comme une variable logique. Par exemple, vous pouvez exécuter le programme :

```
100 REMark variables logiques
110 LET affamé = 1
120 IF affamé THEN PRINT "Prendre un petit pain"
```

Vous attendez une variable logique en ligne 120, mais c'est la variable numérique affamé qui est à sa place. Le système interprétera sa valeur 1 comme vraie et la sortie sera :

```
prendre un petit pain
```

S'il y avait eu en ligne 110 :

```
110 LET affamé = 0
```

il n'y aurait rien eu en sortie. Le système interprète zéro comme faux et toutes les autres valeurs comme vraies.

Ceci est pratique, mais vous pouvez aussi écrire :

```
100 REMark variables logiques
110 LET vrai = 1 : faux = 0
120 LET affamé = vrai
120 IF affamé THEN PRINT "Prendre un petit pain"
```

VARIABLES CHAINES

Il y a beaucoup à dire au sujet des manipulations de chaînes et des variables chaînes, et ceci sera vu dans un autre chapitre.

PROBLEMES SUR LE CHAPITRE 9

1. Un riche marchand de pétrole joue à pile ou face de la manière suivante : pile = 1 ; face : il joue à nouveau, mais son gain est doublé, et ainsi de suite, comme indiqué ci-dessous :

   COUPS 1 2 3 4 5 6 7
   GAINS 1 2 4 8 16 32 64

   En simulant ce jeu, essayez de trouver le montant gagné dans chacun des deux cas :

   a) si le joueur est limité à un maximum de 7 coups par jeu ;

   b) s'il n'y a pas de nombre maximum de coups.

2. Bill et Ben jouent ainsi : à chaque signal, chacun divise son argent en deux parties égales, et en donne la moitié à l'autre. Puis chacun partage en deux son nouveau total, et en passe la moitié à l'autre. Montrer ce qui se passe si Bill démarre avec 16 centimes et Ben avec 64 centimes.

3. Que se passe-t-il si on change le jeu de la sorte que chacun rende à l'autre la moitié de la somme qu'il possède ?

4. Ecrire un programme qui forme des mots de trois lettres aléatoires choisies parmi A,B,C,L) et les écrit jusqu'à l'apparition de 'BAD'.

5. Modifier le dernier programme de sorte qu'il se termine dès que l'un des mots de trois lettres aléatoires est déjà apparu.

CHAPITRE 10

LOGIQUE

Si vous avez lu les précédents chapitres, vous admettez certainement que les répétitions, les décisions, et le partage des tâches en sous-tâches sont les concepts majeurs des problèmes d'analyse et de programmation. Deux de ces concepts, répétition et décision, ont besoin d'expressions logiques comme les suivantes :

```
IF score = 7 THEN EXIT coups
IF suit = 3 THEN PRINT "pique"
```

La première permet une sortie de la boucle **REPeat**. La seconde est simplement une décision de faire ou non quelque chose. Une expression mathématique évalue une expression numérique parmi des millions de valeurs possibles. De la même façon, une expression chaîne peut être évaluée parmi des millions de chaînes de caractères possibles. Vous pouvez alors trouver bizarre que les expressions logiques ne puissent évaluer que l'une des deux valeurs possibles : vrai ou faux.

Dans le cas suivant :

```
score = 7
```

expression évidemment correcte, ou le score = 7 ou non. L'expression est vraie ou fausse. Il se peut qu'à un moment donné, on ne connaisse pas la valeur de l'expression logique, mais elle vient ultérieurement.

AND

Vous devez aussi être très attentif à des expressions telles que **OR**, **AND**, **NOT** qui sont essentielles pour une bonne programmation.

Le mot **AND** en SuperBASIC signifie "et" en Français. Observez le programme suivant :

```
100 REMark AND
110 PRINT "Entrer deux valeurs" \ "1 = VRAI et 0 = FAUX"
120 INPUT pluvieux, trou_dans_le_toit
130 IF pluvieux AND trou_dans_le_toit THEN PRINT "Tout mouillé"
```

Comme dans la vie réelle, vous ne serez mouillé que s'il pleut et s'il y a un trou dans le toit. Si l'une (ou les deux) variable logique :

pluvieux
trou_dans_le_toit

est fausse, alors l'expression logique composée

pluvieux **AND** *trou_dans_le_toit*

est fausse. Il faut deux valeurs vraies pour que l'expression soit vraie. Les règles ci-dessous énoncent ces propriétés. C'est seulement quand l'expression conjuguée est vraie que vous serez mouillé.

pluvieux	trou_dans_le_toit	pluvieux AND trou_dans_le_toit	effet
Faux	Faux	Faux	sec
Faux	Vrai	Faux	sec
Vrai	Faux	Faux	sec
Vrai	Vrai	Vrai	mouillé

Règles pour AND

OR

Dans la vie de tous les jours, le mot "ou" est utilisé de 2 façons. Le ou inclusif et le ou exclusif. Une expression composée utilisant le ou inclusif est vraie si une ou bien les deux expressions qui la composent sont vraies. Essayez le programme suivant :

```
100 REMark OR test
110 PRINT "Enter two values" \ "1 for TRUE or 0 for FALSE"
120 INPUT "Can you bat?", batsman
130 INPUT "Can you bowl?", bowler
140 IF batsman OR bowler THEN PRINT "In the team"
```

Les résultats des différentes combinaisons pissibles figurant dans le tableau suivant :

batsman	bowler	batsman OR bowler	effet
FAUX	FAUX	FAUX	non
FAUX	VRAI	VRAI	oui
VRAI	FAUX	VRAI	oui
VRAI	VRAI	VRAI	oui

Règles pour OR

Avec le OU inclusif, la valeur vrai dans l'une des expression produira la valeur vrai pour l'expression composée.

Si vous écrivez 0 pour faux, et 1 pour vrai, vous obtenez les combinaisons possibles de la numération binaire.

00
01
10
11

NOT ET PARENTHESES

Le mot **NOT** a un sens évident :

NOT vrai = faux
NOT faux = vrai

Cependant, il faut faire attention. Prenons un triangle rouge, et disons :

NOT rouge **AND** carré

cela peut paraître ambigu. Si vous voulez dire :

(**NOT** rouge) **AND** carré

alors pour un triangle rouge, l'expression est fausse.

Si vous écrivez :

NOT(rouge **AND** carré)

pour un triangle rouge, l'expression est vraie. Il faut une règle pour préciser ceci, la voici :

NOT est prioritaire sur **AND**, ainsi l'interprétation de :

(**NOT** rouge) **AND** carré

est la bonne. L'expression est la même que : **NOT** rouge **AND** carré

Pour exprimer les autres interprétations, il faut employer des parenthèses. Dans une expression logique complexe, il est préférable de mettre des parenthèses avec **NOT**, mais si vous le souhaitez, vous pouvez toujours enlever les parenthèses en utilisant les lois suivantes (attribuées à De Morgan)

NOT(a **AND** b) = **NOT** a **OR NOT** b
NOT(a **OR** b) = **NOT** a **AND NOT** b

par exemple :

NOT (grand **AND** beau) = **NOT** grand **OR NOT** beau
NOT(affamé **OR** assoiffé) = **NOT** affamé **AND NOT** assoiffé

Testez ce programme :

```
100 REMark NOT et parenthèses
110 PRINT "Entrez deux valeurs"\"1 pour VRAI ou 0 pour FAUX"
120 INPUT "grand "; grand
130 INPUT "beau "; beau
140 IF NOT (grand AND beau) THEN PRINT "PREMIER"
150 IF NOT grand OR NOT beau THEN PRINT "SECOND"
```

Quelles que soient les combinaisons de nombres entrés, la sortie sera toujours ou deux mots, ou pas du tout, mais pas un. Ceci vous montre que les deux expressions logiques composées sont équivalentes.

XOR OU EXCLUSIF

Supposons qu'un moniteur de golf demande à l'un des assistants ou bien de courir au magasin, ou bien de donner une leçon. Celui-ci fera l'une ou l'autre des deux actions, mais ne pourra faire les deux. Ceci est la situation du ou exclusif. Le programme suivant testerait les candidats:

```
100 REMark XOR test
110 PRINT " Tapez 1 pour oui, 0 pour non."
120 INPUT " Pouvez-vous courir au magasin ?", magasin
130 INPUT " Pouvez-vous enseigner le golf?", golf
140 IF magasin XOR golf THEN PRINT "Acceptable"
```

Les seules combinaisons de réponses qui donnent en sortie "Acceptable" sont (0 et 1) ou (1 et 0). Les règles de **XOR** sont énoncées ci-dessous

Capable de courir au magasin	Capable d'enseigner	magasin XOR golf	Effet
FAUX	FAUX	FAUX	Pas de travail
FAUX	VRAI	VRAI	Obtient l'emploi
VRAI	FAUX	VRAI	Obtient l'emploi
VRAI	VRAI	FAUX	Pas de travail

Règles pou XOR

L'ordre des priorités des expressions logiques est (en commençant par la plus prioritaire)

NOT
AND
OR,XOR

Par exemple l'expression :

riche **OR** *grand* **AND** *beau*

est la même chose que :

riche **OR** *(grand* **AND** *beau)*

L'opération **AND** est effectuée la première. Pour bien montrer que les deux expressions logiques ont le même effet, exécuter le programme suivant :

```
100 REMark Priorités
110 PRINT "Entrez trois valeurs"\"Tapez 1 pour Oui et 0 pour Non"!
120 INPUT riche,grand,beau
130 IF riche OR grand AND beau THEN PRINT "OUI"
140 IF riche OR (grand AND beau) THEN PRINT "AÏE"
```

Quelles que soient les combinaisons des zéros et des uns que vous entrez en ligne 120, vous aurez en sortie ou rien ou

```
OUI
AÏE
```

Vous pouvez vous assurer d'avoir tenté toutes les possibilités en essayant successivement les données formées par tous les nombres binaires de trois chiffres, de 000 à 111.

```
000 001 010 011 100 101 110 111
```

EXERCICES SUR LE CHAPITRE 10

1. Placez dix nombres dans une instruction **DATE**. Lisez chaque nombre et s'il est supérieur à 20, imprimez-le.

2. Testez tous les nombres de 1 à 100 et n'affichez que les carrés parfaits ou les multiples de 7.

3. Des jouets sont décrits de la manière suivante :
   dangereux = D non dangereux = N Chers = C Bon marché = B
   pour les filles = F pour les garçons = G Unisexe = U

   Placez cinq triplets de ces valeurs dans une instruction **DATA** et imprimez uniquement ceux qui sont non dangereux et conviennent aux filles.

4. Modifiez le programme 3 pour n'afficher que ceux qui sont chers et dangereux.

5. Modifiez le programme 3 pour exprimer ceux qui sont non dangereux, non chers et unisexe.

CHAPITRE 11

MANIPULATION DE CHAINES

Vous avez utilisé des variables chaînes pour charger des chaînes de caractères et vous savez que les règles de manipulation de variables chaînes ou de chaînes de constantes ne sont pas les mêmes que celles des variables ou constantes numériques. SuperBASIC met à votre disposition un grand nombre de facilités de manipulations de chaînes de caractères. En particulier, le concept de sous-chaîne étend, et, en même temps, simplifie le problème du découpage d'une chaîne.

ASSIGNATION DE CHAINES

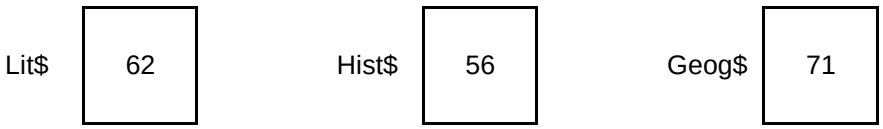
Le chargement d'une variable chaîne réserve autant de caractères qu'il en faut pour la variable. Par exemple :

```
100 LET mots$ = "LONG"
110 LET mots$ = "LONGUEUR"
120 PRINT mots$
```

PRINT affichera le mot de 8 caractères « LONGUEUR ». La première instruction n'aura réservé l'emplacement que de quatre caractères en mémoire, mais elle a été écrasée par la seconde ligne qui réserve huit caractères. Toutefois il est possible d'affecter une dimension à une variable chaîne ; dans ce cas, la longueur maximale de la variable est alors définie, et elle se traite comme un tableau.

CONCATENATION DE CHAINES

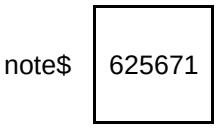
On a quelquefois besoin de construire des enregistrements à partir de plusieurs éléments. Soit, par exemple, les notes de trois matières enseignées : littérature, histoire, géographie. Les notes sont contenues dans les variables :



On peut être amené à créer une seule note de six caractères appelé notes$ on écrira simplement :

```
LET note$ = lit$ & hist$ & geog$
```

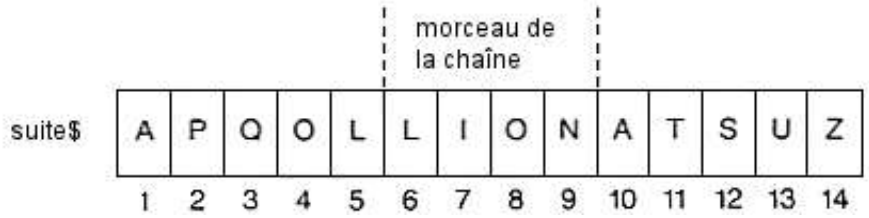
Vous créez ainsi une nouvelle variable :



Remarquons toutefois que nous considérons ici une chaîne de caractères qui peut être numérique, mais note$ n'est pas une variable numérique.

COPIE D'UNE SOUS-CHAINE

Une sous-chaîne est une partie de chaîne. Elle peut contenir de un caractère à la totalité de la chaîne. Pour identifier une sous-chaîne, il faut connaître la position des caractères qui la composent. Soit un jeu d'enfant qui consiste à reconnaître un mot contenu dans une suite de lettres. Chaque lettre a un numéro (un index) indiquant sa position dans la chaîne. La suite de lettres est suite$ et l'indication est : " un chat géant".



Il est clair que la réponse est contenue dans le mot formé par les caractères d'index 6 à 9. On peut écrire :

```
100 suite$ = "APQOLLIONATSUZ"
110 LET réponse$ = suite$(6 TO 9)
120 PRINT réponse$
```

La réponse sera Lion

REMPLACER UN CARACTERE

Maintenant on se propose de remplacer le nom de cet animal par "rat". On peut écrire :

```
130 LET suite$(6 TO 9) = "RAT"
140 PRINT suite$
```

L'exécution de ces cinq lignes de programme provoquera l'affichage suivant :

```
LION
APQOLRATATSUZ
```

Toutes les variables chaînes sont initialement vides, elles ont une longueur nulle. Si on veut copier une chaîne dans une autre de taille insuffisante, SuperBASIC ne prendra pas l'ordre en compte. Vous avez donc intérêt à vous assurer que la chaîne réceptrice sera assez longue en réservant un certain nombre d'espaces, avant d'écrire l'instruction de remplacement :

```
100 LET sujet$ = "ENGLISH MATHS COMPUTING"
110 LET étudiant$ = "     "       "              "
120 LET étudiant$(9 TO 13) = sujet$(9 TO 13)
```

Nous dirons que "RAT" est une sous-chaîne de la chaîne "APQLRATATSUZ" Elle est définie par l'expression :

```
(6 TO 9)
```

qui est appelé facteur de découpage. Il y a d'autres applications. Soulignons que l'instruction **LET** peut être utilisée de diverses façons. Si l'on veut prendre, ou remplacer un caractère unique, on peut écrire :

```
suite$(6 TO 6)
```

qui donnera "R"

On peut remplacer cela par :

```
jumble$(6)
```

Reprenons la variable chaîne note$ "625671". On peut extraire la sous-chaîne indiquant la note d'histoire de la manière suivante :

```
100 LET note$ = "625671"
110 LET hist$ = note$(3 TO 4)
```

Remarquons bien que le résultat de hist$ est la variable chaîne "56". Si vous voulez multiplier son contenu par 1.125 par exemple, il faut que la valeur de hist$ soit d'abord convertie en une donnée numérique, et Super-BASIC se charge lui-même de cette conversion.

```
120 LET nombre = 1.125 * hist$
```

La ligne 120 va convertir "56" en 56 et le multiplier par 1.125 pour donner 63.

Problème inverse : convertissons le nombre 63 en chaîne '63'. A nouveau, SuperBASIC se chargera de cette conversion. Si nous entrons :

```
130 LET note$(3 TO 4) = nombre
140 PRINT note$
```

Vous aurez en sortie 626371 qui montre que la note d'histoire est remplacée par 63.

A strictement parler, il est incorrect de mélanger les types de données dans une instruction LET. II serait stupide d'écrire :

```
LET nombre = "LION"
```

et vous auriez un message d'erreur si vous tapiez :

```
LET nombre = "65"
```

Le système comprendrait que vous voulez mettre dans nombre la valeur numérique 65, et ferait ainsi. Le programme complet sera :

```
100 LET note$ = "625671"
110 LET hist$ = note$(3 TO 4)
120 LET nombre = 1.125 * hist$
130 LET note$(3 TO 4) = nombre
140 PRINT note$
```

et le résultat sera 63.

A la ligne 120, le système a converti la valeur d'une chaîne en nombre, pour pouvoir la multiplier. A la ligne 130, un nombre a été converti en chaîne. Cette technique est connue sous le nom de **conversion de type**.

On peut écrire le programme plus simplement si l'on connaît à la fois les sous-chaînes et les conversions de types.

```
100 LET note$ = "625671"
110 LET note$(3 TO 4) = 1.125 * note$(3 TO 4)
120 PRINT note$
```

Si vous connaissez d'autres BASIC, vous apprécierez la simplicité et la puissance des sous-chaînes et de la conversion de type.

RECHERCHE DANS UNE CHAINE

On peut rechercher une sous-chaîne dans une chaîne. Le programme suivant affiche une suite de lettres et vous invite à deviner l'animal dont le nom est une sous-chaîne de cette suite.

```
100 REM Cherche l'animal
105 CLS : DIM an$(3)
110 LET suite$ = "CRANE PREHISTORIQUE"
120 PRINT SUITE$
130 INPUT "Quel est l'animal?" ! an$
140 IF an$ INSTR suite$ AND an$(1) = "A"
150   PRINT "Correct"
160 ELSE
170   PRINT "Incorrect"
180 END IF
```

La fonction **INSTR** renvoie zéro si on n'a pas deviné juste, sinon, elle renvoie le nombre qui représente la position de départ de la sous-chaîne, ici 6.

Parce que l'expression

```
INSTR(suite$,an$)
```

peut être considérée comme une expression logique, si la recherche est couronnée de succès, l'expression est considérée comme juste, alors que si la recherche montre que la sous-chaîne n'est pas contenue dans la chaîne, l'expression sera considérée comme fausse.

AUTRES FONCTIONS CHAINES

Vous avez déjà rencontré **LEN** qui renvoie la longueur (ou le nombre de caractères d'une chaîne). Il est possible de reproduire plusieurs fois une chaîne ou un caractère particulier. Par exemple, si vous voulez écrire en sortie une suite d'astérisques, plutôt que d'en taper 40, dans une instruction PRINT, ou dans une boucle, vous pouvez simplement écrire :

```
PRINT FILL$ ("*",40)
```

Enfin, il est possible de convertir les valeurs du code ASCII en chaînes de caractères, à l'aide de l'instruction CHR$

Par exemple :

```
PRINT CHR$(65)
```

donnera en sortie la lettre A.

COMPARAISON DE CHAINES

Pour comparer deux chaînes, on compare d'abord leurs premiers caractères. Les lettres A,B,C... étant codées 65,66,67... il est naturel que les expressions suivantes soient correctes :

A plus petit que B
B plus petit que C

La machine comparant caractère par caractère, on aura :

CAT plus petit que DOC
CAN plus petit que CAT

Vous pouvez par exemple écrire :

```
IF "CAT" < "DOG" THEN PRINT "MIAOU"
```

et vous aurez en sortie : `MIAOU`

De la même façon :

```
IF "DOG" > "CAT" THEN PRINT "WOOF"
```

donnera en sortie : `WOOF`

Nous utilisons les symboles mathématiques de comparaison pour les comparaisons de chaînes. Toutes les expressions logiques suivantes sont à la fois autorisées et vraies.

```
"ALF" < "BEN"
"KIT" > "BEN"
"KIT" <= "LEN"
"KIT" >= "KIT"
"PAT" >= "LEN"
"LEN" <= "LEN"
"PAT" <> "PET"
```

Les comparaisons basées sur les codes internes de la machine ont un sens, mais les données ne se limitent pas aux caractères majuscules. Nous nous attendons par exemple à ce que :

Cat soit plus petit que COT et
K2N plus petit que K27N

Une comparaison caractère par caractère basée sur les codes internes ne donnera pas ces résultats, aussi SuperBASIC procède-t-il d'une manière plus intelligente. Le programme suivant, qui demande deux entrées et fournit le résultat, illustre les règles de comparaison de chaînes.

```
100 REMark comparaisons
110 REPeat comp
120   INPUT "Entrez une chaine de caractères" ! first$
130   INPUT "Entrez une autre chaine de caractères" ! second$
140   IF first$ < second$ THEN PRINT "Plus petit"
150   IF first$ > second$ THEN PRINT "Plus grand"
160   IF first$ = second$ THEN PRINT "Egales"
170 END REPeat comp
```

ENTREE		SORTIE
CAT	COT	Plus petit
CAT	CAT	Egales
PET	PETE	Plus petit
K6	K7	Plus petit
K66	K7	Plus grand
K12N	K6N	Plus grand

> Plus grand que – La comparaison dépend de la casse – Les nombres sont comparés selon leur ordre

< Plus petit que – La comparaison dépend de la casse – Les nombres sont comparés selon leur ordre

= Egal - – La comparaison dépend de la casse – Les chaines de caractères doivent être les mêmes

== Très peu différent – Les chaines sont partiquement identiques. La comparaison dépend de la casse – Les nombres sont comparés selon leur ordre

>= Supérieur ou égal à – La comparaison dépend de la casse – Les nombres sont comparés selon leur ordre

<= inférieur ou égal à Case – La comparaison dépend de la casse – Les nombres sont comparés selon leur ordre

EXERCICES SUR LE CHAPITRE 11.

1. Placer douze lettres, toutes différentes, dans une variable chaîne, puis construisez une autre variable chaîne de 6 lettres. Faire une recherche de six lettres, chacune à son tour dans la 1ère chaîne, et indiquer dans chaque cas, si la lettre est contenue ou non dans la chaîne.

2. Refaire l'exercice 1 en utilisant un tableau de caractères. Placer 20 majuscules aléatoires et afficher celles qui apparaissent deux ou plusieurs fois.

3. Ecrire un programme qui lit une partie d'un texte écrit en caractères majuscules. Calculer la fréquence de chaque lettre et afficher les résultats

   "GOUVERNMENT IS A TRUST, AND THE OFFICERS OF THE GOVERNMENT ARE TRUSTEES, AND BOTH THE TRUST AND THE TRUSTEES ARE CREATED FOR THE BENEFIT OF THE PEOPLE. HENRY CLAY. 1829."

4. Ecrire un programme qui compte le nombre de mots du texte suivant. On reconnait un mot parce qu'il commence par une lettre et se termine par un espace ou un point.

   "THE REPORTS OF MY DEATH ARE GREATLY EXAGERATED. CABLE FROM MARK TWAIN TO THE ASSOCIATED PRESS, LONDON, 1896."

5. Réécrire le dernier programme en utilisant des procédures et des variables logiques.

CHAPITRE 12

SORTIES ECRAN

SuperBASIC a tellement étendu la valeur et la variété des facilités de la présentation de l'écran que nous décrivons ses caractéristiques en deux parties : simple affichage et écran.

La première partie décrit la sortie d'un texte ordinaire. Nous expliquons ici les méthodes élémentaires d'affichage de messages, textes ou sorties numériques. Même dans cette partie banale, il y a des innovations dans la notion « d'espace intelligent », un moyen d'allier facilité de saisie et commodité d'interprétation.

La seconde partie est plus importante. Elle montre la grande facilité des constructions. Par exemple, il est possible de tracer un cercle en écrivant tout simplement le mot **CIRCLE** suivi de quelques détails qui définissent la position et sa taille. Dans d'autres systèmes, il faut avoir de bonnes notions de géométrie et de trigonométrie pour comprendre des notions, somme toute, très simples.

Nous remarquerons que chaque mot-clé a été attentivement choisi pour exprimer la notion qu'il évoque :

WINDOW (fenêtre) : définit une surface de l'écran,
BORDER (bordure) : construit une bordure tout autour,
PAPER (papier) : définit la couleur du fond,
INK (encre) : détermine la couleur de ce que vous écrivez sur le fond.

Avec un peu d'effort et de pratique, vous vous souviendrez vite des noms des mots-clés et de ce qu'ils font. Vous programmez ainsi de mieux en mieux. Et avec l'expérience, vous constaterez que les graphiques peuvent produire une nouvelle forme d'art.

AFFICHAGE SIMPLE

Le mot-clé **PRINT** peut être suivi par une suite d'items. Un **PRINT** item peut être :

- un texte comme "ceci est un texte"
- des variables comme : nombres, mot$
- des expressions comme : 3*nombre, ou jour$ & semaine$

Les **PRINT** items peuvent être mélangés dans toute instruction **PRINT**, mais il faut un séparateur entre deux items consécutifs. Les séparateurs de **PRINT** peuvent être :

; aucun autre effet que celui de séparer deux items
! Espace "intelligent", insère en principe un espace entre deux items, sauf si le second item est un début de la ligne suivante.
, La sortie aura lieu à la tabulation suivante, soit au début de la zone suivante de 8 caractères.
**** Passage à la ligne suivante.

Les nombres 1,2,3 sont des print items pratiques pour illustrer les effets des séparateurs print.

Instruction	Effet
`100 PRINT 1,2,3`	`1  2  3`
`100 print 1 ! 2 ! 3 !`	`1 2 3`

`100 PRINT 1 \ 2 \ 3`	`1`
	`2`
	`3`

`100 PRINT 1 ; 2 ; 3`	`123`
`100 PRINT "Ceci est du texte"`	`Ceci est du texte`
`100 LET word$ = " "` `110 PRINT word$`	Moves print position
`100 LET nombre = 13` `110 PRINT nombre`	`13`
`100 LET an$ = "oui"` `110 PRINT "Je dis" ! an$`	`Je dis oui`
`110 PRINT"La somme fait" ! 4+2`	`La somme fait 6`

PROBLEME DE PRESENTATION

Avec la commande **AT**, vous pouvez positionner les données en sortie en n'importe quel endroit de la ligne courante ou de l'écran. Par exemple

```
AT 10,15 : PRINT " Ceci est en colonne 10 et ligne 15"
```

On peut aussi utiliser la commande **CURSOR** pour positionner les données en sortie en n'importe quel endroit de l'écran qui devient une grille de 256, 512 pixels. Par exemple :

```
CURSOR 100,150 : PRINT " ceci est un point d'abscisse 100 et d'ordonnée 150."
```

La lecture du *Guide des Mots-clés* peut vous paraître beaucoup plus difficile à comprendre que la description ci-dessus. Deux grandes difficultés disparaîtront si vous retenez ceci :

- Les expressions les plus simples sont : texte entre guillemets, variables, nombres.
- Un print item nécessite un print séparateur.

ECRAN

Cette partie présente les commandes qui permettent les sorties sous forme de texte ou de graphiques. L'instruction

```
MODE 8 ou MODE 256
```

sélectionnera le Mode 4 dans lequel il y a :

- 512 pixels en abscisse numérotés de 0 à 511
- 256 pixels en ordonnée numérotés de 0 à 255
- 4 couleurs

COULEUR

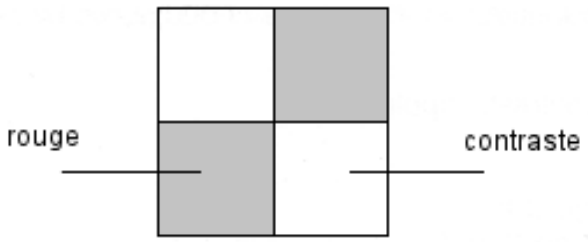
On sélectionne une couleur en utilisant le code suivant avec les mots-clés qui conviennent tels que **PAPER**, **INK**, etc... Remarquez bien que les nombres tout seuls ne signifient rien, ils ne peuvent être interprétés comme des couleurs que s'ils sont utilisés en conjonction avec les mots--clés **PAPER**, **INK**...

Codes couleurs :

Mode 8	Code	Mode 4
Noir	0	Noir
Bleu	1	Noir
Rouge	2	Rouge
Magenta	3	Rouge
Vert	4	Vert
Cyan	5	Vert
Jaune	6	Blanc
Blanc	7	Blanc

Par exemple INK 3 donnerait magenta en MODE 8.

Il est possible de spécifier deux couleurs dans une instruction. Par exemple : 2,4 donne un carré en damier (trame) comme indiqué ci-dessous. Dans chaque groupe de 4 pixels, il y en aurait deux rouges, correspondant à la couleur sélectionnée en premier. Les deux autres pixels seront un contraste. Mais ces effets ne sont guère visibles sur un téléviseur.



si vous écrivez :

```
INK 2,4
```

la figure sera formée des couleurs de codes 2 et 4, que nous appellerons couleur et contraste.

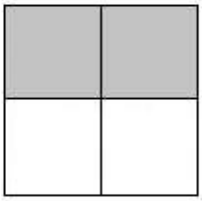
TRAMES

Amusez-vous à faire quelques essais avec les trames, puis observez les détails complémentaires indiqués ci-dessous :

```
100 REMark couleur/contraste
110 FOR couleur = 0 TO 7 STEP 2
120   PAPER couleur : CLS
140   FOR contraste = 0 TO 7 STEP 2
150     BLOCK 100,50,40,50,couleur,contraste
160     PAUSE 50
170   END FOR contraste
180 END FOR couleur
```

AUTRES TRAMES

Si vous désirez obtenir différentes figures, vous pouvez ajouter un troisième numéro de code aux couleurs déjà sélectionnées. Par exemple :



Les symboles suivants correspondent à Rouge Contraste

Code	Nom	Effet
0	1 seul carré en contraste	
1	Deux carrés horizontaux	
2	Deux carrés verticaux	
3	En damier	

PARAMETRES DES COULEURS

Il y a trois paramètres numériques pour les couleurs et les trames qui sont :

```
INK colour, contrast, stipple
```

et peuvent être utilisés dans les limites ci-dessous :

- couleur : paramètre compris entre 0 et 7
- contraste : paramètre compris entre 0 et 7
- trame : paramètre compris entre 0 et 3

Il existe un autre moyen d'obtenir le même résultat avec un seul paramètre. Voyez pour cela le « *Guide de référence des concepts* ».

Le programme suivant vous indique comment afficher tous les effets possibles de couleurs :

```
100 REMark Effets couleurs
110 FOR nombre = 0 TO 255
120   BLOCK 100,50,40,50,nombre
130   PAUSE 50
140 END FOR nombre
```

COULEUR DE FOND

La couleur du fond s'exprime par l'instruction **PAPER** suivie d'un, de deux ou de trois paramètres. Par exemple :

```
PAPER 2       {rouge}
PAPER 2,4     { damier rouge et vert }
PAPER 2,4,1   { carrés horizontaux en rouge et vert }
```

Pour que ces couleurs soient visibles, il faut d'abord que l'écran soit nettoyé avec la commande **CLS**.

COULEUR DE L'ENCRE

INK suivi de un, deux ou trois paramètres indique les couleurs d'impression des caractères, lignes et figures graphiques. L'instruction **INK** se comporte comme les couleurs. Par exemple :

```
INK 2         { caractères ou traits en rouge }
INK 2,4       { caractères ou traits en damier rouge et vert }
INK 2,4,1     { caractères ou traits rouge et vert en carrés horizontaux }
```

La couleur des traits sera changée à chaque changement de carré.

CLS

CLS signifie nettoyer le fond de la fenêtre dans laquelle on travaille, comme on efface un tableau ; mais ici le tableau est électronique et multi-couleurs.

FLASHING - CLIGNOTEMENT

Vous ne pouvez faire clignoter la couleur des traits que dans le mode 8. Pour ce faire, il suffit de taper

```
FLASH 1
```

et pour interrompre le clignotement :

```
FLASH 0
```

L'effet de clignotement permet d'attirer l'attention sur le texte qui clignote.

FICHIERS

Vous avez déjà utilisé les microdrives pour les chargements des programmes et vous avez déjà utilisé les instructions **LOAD** et **SAVE**. On peut charger sur cartouche des données et des programmes. Le mot « fichier » signifie habituellement un ensemble d'enregistrements de données et de programmes. On appelle enregistrement une suite d'informations se rapportant au même article, ou au même individu, comme le nom, l'adresse et le numéro de téléphone.

Deux des types de fichiers les plus utilisés sont les fichiers séquentiels et les fichiers à accès direct. Dans un fichier séquentiel les enregistrements sont lus en séquence à partir du premier. Pour accéder au cinquantième il faut d'abord lire les quarante-neuf enregistrements qui précèdent, alors que le cinquantième enregistrement d'un fichier à accès direct est accédé directement et immédiatement, le système n'ayant pas à parcourir les enregistrements précédents. Les fichiers séquentiels fonctionnent comme les cassettes musicales, tandis qu'un disque longue durée contenant huit morceaux est du même type que les fichiers à accès direct : on peut positionner le bras de lecture directement sur l'un des huit morceaux.

Le type de fichiers le plus simple est une suite de nombres. On peut construire, par exemple, les nombres 1 à 100 dans un fichier appelé nombres. Le nom du fichier doit toutefois comporter deux parties :

- le nom du périphérique,
- le contenu du fichier.

Créons le fichier « nombres », sur la cartouche située dans le microdrive 1, dont le nom est

```
mdv1_
```

et le contenu :

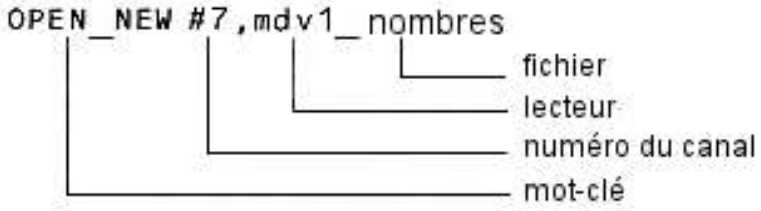
```
nombres
```

Aussi le nom complet du fichier sera

```
mdv1_nombres
```

CANAUX

Un programme peut utiliser plusieurs fichiers à la fois, et on se réfère plus commodément à un fichier par le numéro du canal associé au fichier, qui est un entier compris entre 0 et 15. On associe un fichier à un numéro de canal à l'aide de l'instruction **OPEN**, ou, s'il s'agit d'un nouveau fichier de l'instruction **OPEN_NEW**. Choisissons, par exemple, le canal numéro 7, nous écrivons :



Par la suite, on peut faire appel au fichier en écrivant simplement #7. Voici un programme complet :

```
100 REMark simple fichier
110 OPEN_NEW #7,mdv1_nombres
120 FOR nombre = 1 TO 100
130   PRINT #7,nombre
140 END FOR nombre
150 CLOSE #7
```

Le numéro 7 ayant été associé au fichier cartouche, l'instruction **PRINT** va provoquer l'écriture des nombres sur la cartouche. L'instruction CLOSE#7 est nécessaire car le système doit exécuter une suite d'actions après l'utilisation du fichier.

De plus, elle libère le canal 7 pour d'autres utilisations éventuelles. Après l'exécution du programme précédent, tapons :

```
DIR mdv1_
```

et la directory (répertoire) affichera à l'écran le nombre de fichiers existant sur la cartouche du Microdrive

Pour lire ce fichier, on emploie le mot-clé **OPEN_IN** ; par ailleurs le programme de lecture de données à partir d'un fichier est le même que ci-dessus.

```
100 REMark Lecture du fichier
110 OPEN IN #6, mdv1_nombres
120 FOR item = 1 TO 100
130 INPUT #6, nombre
140 PRINT ! nombre !
150 END FOR item
```

```
160 CLOSE #6
```

On aura en sortie les nombres de 1 à 100, mais bien sûr, à condition que la cartouche qui contient le fichier nombres soit encore dans le Microdrive mdv1_

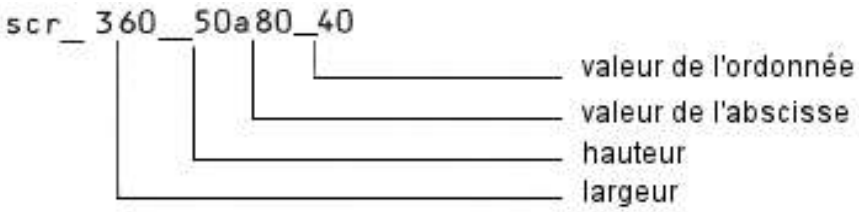
PERIPHERIQUES ET CANAUX

Vous venez de voir un exemple d'utilisation d'un périphérique : un fichier de données sur un microdrive. Ainsi, non seulement un fichier a été ouvert, mais, de plus, on a associé un périphérique à un canal particulier. Certains périphériques ont des canaux qui leur sont associés en permanence :

Canal	Utilisation
#0	SORTIE – fenêtre de commande ENTREE – clavier
#1	SORTIE – affichage dans une fenêtre de l'écran
#2	SORTIE – liste des programmes à l'écran

On peut créer une fenêtre de n'importe quelle taille où l'on veut sur l'écran. On utilise pour cela la commande :

```
scr
```



Le programme suivant crée une fenêtre associée au canal numéro 5, la colore en rouge (code 2) et donne l'ordre de fermeture :

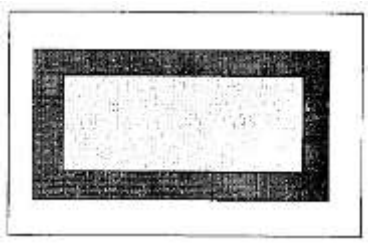
```
100 REMark Création d'une fenêtre
110 OPEN #5, scr_400x200a20x50
120 PAPER #5,4 : CLS #5
130 CLOSE #5
```

Remarquez bien que chaque fenêtre peut avoir ses propres paramètres, tels que **PAPER**, **INK**, etc... Une fenêtre qui a été ouverte n'est plus la fenêtre prise par défaut par le système.

On peut changer la position ou la forme d'une fenêtre, avant sa fermeture, sans avoir à la fermer et à la rouvrir. Essayez d'ajouter deux lignes au programme précédent :

```
124 WINDOW #5,300,100,110,65
126 PAPER #5,2 : CLS #5
```

Exécutez à nouveau le programme, vous verrez alors une fenêtre verte à l'intérieur de la fenêtre rouge qui était là précédemment. Cette fenêtre verte est maintenant associée au canal 5. Voir la figure ci-dessous :



BORDURE

On peut construire une bordure autour de l'écran ou d'une fenêtre. Par exemple :

```
BORDER #5,6
```

crée une bordure autour de la fenêtre associée au canal 5. Elle aura une épaisseur de 6 unités, et la taille de la fenêtre sera réduite d'autant. La bordure peut être transparente. On peut aussi lui attribuer une couleur par la méthode habituelle :

```
BORDER #5,6,2
```

donnera une bordure rouge. On peut construire des bordures d'autres couleurs et compositions par les méthodes habituelles. Par exemple :

```
BORDER 10
```

ajoute 10 pixels à l'épaisseur de la bordure transparente de la fenêtre en cours (transparente car il n'a pas été spécifié de couleur) et :

```
BORDER 2,0,7,0
```

ajoute 2 pixels à l'épaisseur d'une bordure en damier blanc et noir.

BLOCK

On peut spécifier la taille, la position et la couleur d'un rectangle en une seule instruction. Il sera construit dans le système de coordonnées relatif à la fenêtre en cours. Par exemple :

```
BLOCK #5,10,20,50,100,2
```

crée un rectangle dans la fenêtre #5, à la position d'abscisse 50 et d'ordonnée 100. Il a une largeur de 10 unités et une hauteur de 20 unités, et la couleur rouge. On peut noter que les instructions **WINDOW** et **BLOCK** s'appliquent de la même façon en mode 4 et en mode 8, (bien que les couleurs varient) parce que les valeurs en abscisse varient de 0 à 511 et de 0 à 211 en ordonnée.

AFFICHAGES SPECIAUX CSIZE

On peut modifier la taille des caractères. Par exemple :

```
CSIZE 3,1
```

donne les plus grands caractères possibles et,

```
CSIZE 0,0
```

donne les plus petits possibles. Le premier nombre varie de 0 à 3 et indique la largeur, le second est zéro ou 1 et il indique la hauteur. Les tailles normales sont :

```
MODE 4    CSIZE 0,0   25 lignes de 84 caractères
MODE 8    CSIZE 2,0   25 lignes de 42 caractères
```

Le nombre de lignes et de colonnes disponibles pour chaque taille de caractères dépendent de l'appareil utilisé. Les valeurs ci-dessus se rapportent au moniteur, celles d'un téléviseur sont plus petites et varient d'un téléviseur à l'autre.

De plus en basse résolution, on ne peut sélectionner une taille de caractères plus petite que celle prise par défaut.

STRIP

On peut obtenir un fond spécial pour faire ressortir les caractères. Par exemple :

```
STRIP 7
```

donne une bande blanche, tandis que :

```
STRIP 2,4,2
```

donne un fond rayé verticalement en vert et rouge. Toutes les combinaisons habituelles de couleurs sont possibles.

OVER

Normalement l'affichage se présente sur la couleur du fond en cours. L'instruction **STRIP** permet de modifier cela, de même que l'instruction **OVER**.

`OVER 1`	1 écrit à la couleur spécifiée par INK sur des bandes transparentes.
`OVER -1`	-1 écrit à la couleur spédifiéeINK par-dessus l'écran existant

Pour revenir à l'affichage rayé précédent :

```
OVER 0
```

Vous pouvez souligner des caractères.

UNDER

Vous pouvez souligner des caractères.

`UNDER 1`	souligne toutes les sorties ultérieures avec la couleur en cours
`UNDER 0`	supprime l'ordre de souligner.

ECHELLES GRAPHIQUES

Pour tracer de vraies figures géométriques sur un téléviseur ou sur un écran vidéo, il n'est guère facile d'utiliser le système pixels. Le système QL vous fournit une méthode pratique pour tracer cercles, carrés et autres formes.

Les coordonnées graphiques admettent par défaut l'échelle 100 en ordonnée, et l'abscisse de votre choix.

L'origine en mode graphique n'est pas le même que l'origine dans le système pixels. Il est en bas à gauche de l'écran ou de la fenêtre en cours.

POINTS ET DROITES

L'échelle graphique permet un tracé aisé de points et de droites. Avec une échelle verticale de 100, un point proche du centre de la fenêtre sera construit ainsi :

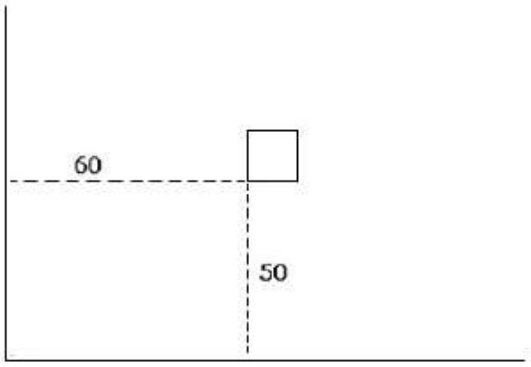
```
POINT 60,50
```

Le point d'abscisse 60 et d'ordonnée 50 sera de la couleur de **INK** en cours. De la même façon, l'instruction **LINE** construit des droites :

```
LINE 60,50 TO 80,90
```

On peut mettre plusieurs droites bout à bout. Par exemple, l'instruction suivante construit un carré

```
LINE 60,50 TO 70,50 TO 70,60 TO 60,60 TO 60,50
```



MODE RELATIF

Un couple de coordonnées telles que abscisse, ordonnée définit un point par rapport à l'origine (0,0) au bas gauche d'une fenêtre. Il est parfois plus pratique de définir des points par rapport à la position du point précédent (donc du curseur). Par exemple, on peut construire le carré ci-dessus d'une autre façon, à l'aide de l'instruction **LINE_R** :

```
POINT 60,50
LINE_R 0,0 TO 10,0 TO 0,10 TO -10,0 TO 0,-10
```

La première instruction **POINT** positionne le point de départ, puis, chaque droite étant tracée, l'extrémité de l'une devient le point de départ de la suivante. Le programme suivant construit une figure formée de carrés de positions et de couleurs aléatoires

```
100 REMark Carrés de couleurs
110 PAPER 7 : CLS
120 FOR cc = 1 TO 100
130   INK RND(1 TO 6)
140   POINT RND(90),RND(90)
150   LINE_R 0,0 TO 10,0 TO 0,10 TO -10,0 TO 0,-10
160 END FOR cc
```

On pourrait obtenir le même résultat avec l'instruction **LINE**, mais il serait beaucoup plus compliqué d'écrire les paramètres de cette instruction.

CERCLES ET ELLIPSES

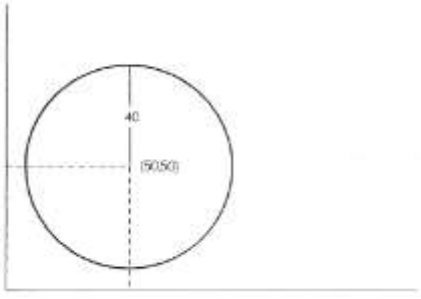
Pour tracer un cercle, il faut spécifier :

- la position du centre (50,50 par exemple)
- le rayon (40 par exemple)

L'instruction :

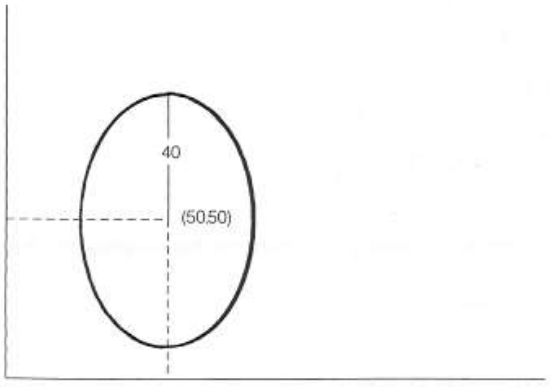
```
CIRCLE 50,50,40
```

construit un cercle dont le centre est à la position 50,50, et le rayon 40, comme sur la figure ci-dessous :



Avec un quatrième paramètre on obtient une ellipse.

```
CIRCLE 50,50,40,.5
```

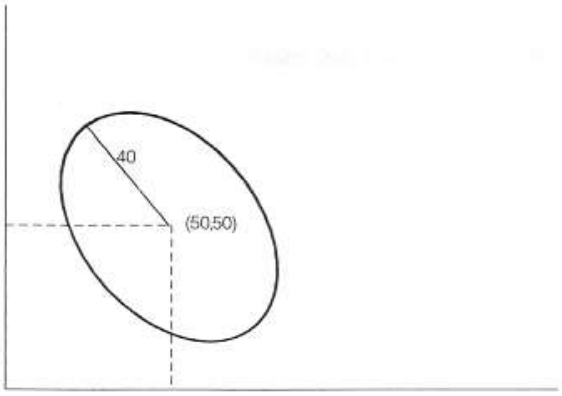


Les mots-clés **CIRCLE** et **ELLIPSE** sont interchangeables.

Le grand rayon de l'ellipse est 40, mais le petit rayon est la moitié du grand. Le nombre 0.5 est appelé le coefficient de réduction du rayon. S'il est égal à 1, la figure est un cercle, s'il est différent de 1, ce sera une ellipse, s'il vaut zéro, la figure sera une droite. Pour incliner l'ellipse, on ajoute un cinquième paramètre, comme ci-dessous :

```
CIRCLE 50,50,40,.5,1
```

qui incline l'ellipse d'un angle égal à 1 radian, soit environ 57°, comme sur la figure suivante :



Traçons une figure d'ellipses avec le programme suivant :

```
100 FOR rot = 0 TO 2*PI STEP PI/6
110   CIRCLE 50,50,40,0.5,rot
120 END FOR rot
```

Voici l'ordre des paramètres pour un cercle ou une ellipse :

abscisse_du_centre, ordonnée_du_ centre, grand_rayon,[coefficient de réduction]angle]

Les deux derniers paramètres sont optionnels, c'est pourquoi ils sont entre crochets ([ ])

Exemple :

Ecrire un programme qui fait :

1. Ouverture d'une fenêtre de 350 sur 180
2. Etablissement de l'échelle 100 en mode 8
3. Sélection d'un fond noir et nettoyage de la fenêtre
4. Construction d'une bordure verte de 2 unités de largeur
5. Tracé d'une figure de six ellipses de couleur.
6. Fermeture de la fenêtre.

```
100 REMark figure
110 MODE 8
120 OPEN #7,scr_350x180a100x50
130 SCALE #7,100,0,0
140 PAPER #7,0 : CLS #7
150 BORDER #7,2,4
160 FOR couleur = 1 TO 6
170   INK #7,couleur
180   LET rot = 2*PI/couleur
190   CIRCLE #7,50,50,30,0.5,rot
200 END FOR couleur
210 CLOSE #7
```

On obtiendra des effets intéressants en modifiant ce programme. Par exemple, essayons les modifications suivantes :

```
160 FOR couleur = 1 TO 100
180 LET rot = couleur*PI/50
```

ARCS

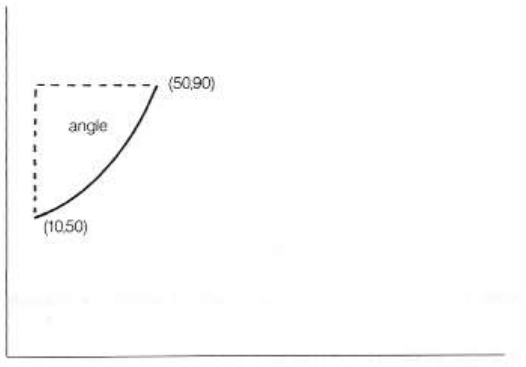
Pour tracer un arc, il faut connaître :

- le point de départ,
- le point d'arrivée,
- la valeur de l'angle qui l'intercepte (en radians).

Essayez par exemple :

```
ARC 10,50 TO 50,90,1
```

qui donne un arc de courbure modérée de la couleur en cours fixée par la dernière instruction **INK**.



FILL

On peut colorer l'intérieur d'une figure fermée avec la couleur **INK**, en écrivant simplement :

```
FILL 1
```

avant de tracer la figure. Le programme suivant construit un cercle vert :

```
INK 4
FILL 1
CIRCLE 50,50,30
```

L'instruction FILL 0 annule l'effet de FILL.

DEFILEMENT ET DEPLACEMENT LATERAL

On peut obtenir, dans une fenêtre, un défilement de l'écran, ou bien d'une image fixe. Le défilement (scroll) s'exprime en pixels. Un défilement vers le haut s'exprime par un nombre positif; ainsi :

```
SCROLL 10
```

déplace le contenu de la fenêtre en cours de 10 pixels vers le haut.

```
SCROLL -8
```

déplace e contenu de 8 pixels vers le bas.

Un second paramètre sert à exprimer un défilement partiel :

```
SCROLL -8, 1
```

fera défiler seulement la partie au-dessus de la ligne contenant le curseur, (ligne du curseur exclue) et :

```
SCROLL -8, 2
```

fera défiler la partie au-dessous de la ligne contenant le curseur (ligne du curseur exclue).

Dans le défilement, l'espace libéré par le déplacement peut être de la couleur du fond. Si vous voulez l'éviter, utilisez 0 en second paramètre.

On peut déplacer le contenu de la fenêtre vers la droite ou vers la gauche. L'instruction **PAN** fonctionne comme l'instruction **SCROLL**, mais

`Pan 40` déplace le contenu vers la droite
`Pan -40` déplace le contenu à gauche

Un second paramètre permet d'obtenir un déplacement partiel :

- 0 tout l'écran
- 3 La totalité de la ligne qui contient le curseur
- 4 La partie de la ligne située à droite du curseur, curseur inclus.

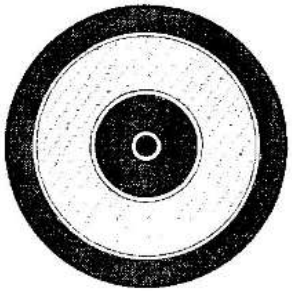
En mode tramé, ou en mode 8, les paramètres de ces instructions doivent être exprimés en multiples de 2 pixels.

EXERCICES SUR LE CHAPITRE 12

1. Faire un programme qui construit une grille de 10 rangées de 10 carrés.

2. Placer les nombres de 1 à 100 dans les carrés, en partant du bas gauche, et mettez un F dans le dernier carré.

3. Tracer une cible de fléchettes sur l'écran, formée d'un anneau extérieur qui peut contenir des nombres, un deuxième et un troisième anneau, comme sur la figure ci-dessous, avec, au centre un cercle entouré d'un anneau.



CHAPITRE 13

TABLEAUX

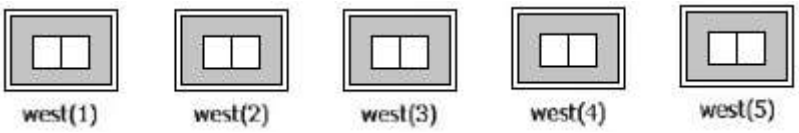
POURQUOI DES TABLEAUX

Supposons que vous êtes directeur de prison, et que vous avez un nouveau bloc appelé le bloc Ouest (West). Il peut contenir 50 prisonniers. Vous devez savoir quel prisonnier (désigné par son numéro) sera dans quelle cellule. On pourrait donner un nom à chaque cellule, mais il est plus simple de lui donner un numéro de 1 à 50.

Dans une simulation informatique, nous ne considérons que 5 prisonniers dont les numéros seront entrés dans une instruction **DATA**.

```
Data 50, 37, 86, 41, 32
```

Nous construisons un tableau de variables qui ait le même nom "West", et que l'on distinguera par le numéro écrit entre parenthèses.



L'instruction **DIM** déclare un tableau et lui affecte une dimension :

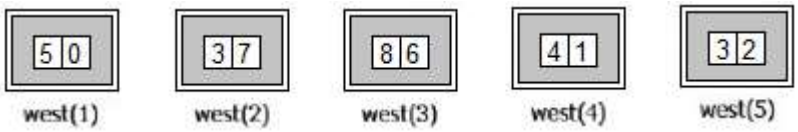
```
DIM west(5)
```

qui permet au SuperBASIC de réserver l'espace nécessaire. Après l'exécution de l'instruction **DIM**, on pourra utiliser les cinq variables.

L'instruction **DATA** permettra d'affecter à chaque cellule le numéro du détenu, par l'instruction **READ**.

```
FOR cellule = 1 TO 5 : READ west (cellule)
```

Ajoutons une autre boucle **FOR** et une instruction **PRINT** pour nous assurer que les numéros sont bien dans les cellules.



Voici le programme complet :

```
100 REMark Prisonniers
110 DIM west(5):RESTORE
120 FOR cellule = 1 TO 5 : READ west(cellule)
130    FOR cellule = 1 TO 5 : PRINT cellule! west(cellule)
140 REMark Fin du programme
150 DATA 50, 37, 86, 41, 32
```

qui donnera en sortie :

```
1     50
2     37
3     86
4     41
```

```
5     32
```

Les nombres de 1 à 5 sont appelés les indices du tableau west. Ce tableau est numérique et il est formé de 5 éléments.

Si on remplaçait la ligne 130 par :

```
130 PRINT west
```

on aurait en sortie :

```
0
50
37
86
41
32
```

Il y a un zéro au début de liste car la valeur des indices commence à 0. Nous verrons par la suite combien cet élément zéro peut être pratique.

Notons aussi que lorsqu'on **DIM**ensionne un tableau, le système place la valeur zéro dans chaque élément.

TABLEAUX DE CHAINES

Les tableaux de chaînes fonctionnent comme les tableaux numériques, mais ils comportent un paramètre supplémentaire qui indique la longueur de chaque variable chaîne du tableau. Supposons que dix joueurs de golf soient désignés par leurs noms dans une instruction DATA

```
DATA "Tom","Graham","Sevvy","Jack","Lee"
DATA "Nick","Bernard","Ben","Gregg","Hal"
```

Il faudra 10 noms de variables différents, mais s'il y avait une centaine ou un millier de joueurs, le travail deviendrait fastidieux, voire impossible. Pour éviter cette sorte de problème, on utilisera un tableau, dans lequel chaque variable aura deux parties :

- un nom conforme aux règles habituelles
- une partie numérique appelée l'indice

On écrira ainsi le nom des variables :

```
flat$(1),flat$(2),flat$(3)...etc
```

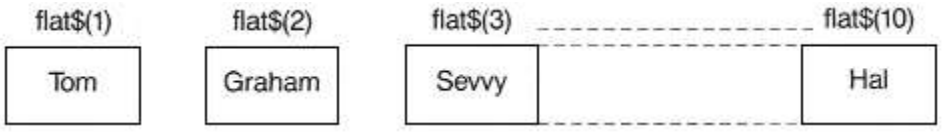
Avant d'écrire les variables du tableau, il faut déclarer le tableau et ses dimensions :

```
DIM flat$(10,8)
```

qui réserve 10 variables à l'usage du programme. Chaque variable chaîne du tableau peut avoir jusqu'à huit caractères. Les instructions **DIM** sont en général placées toutes ensemble au début de programme. Une fois que le tableau a été déclaré par une instruction **DIM**, on peut en utiliser tous les éléments. L'un des avantages importants de cette méthode est la possibilité d'utiliser l'indice comme une variable numérique. On peut en effet écrire :

```
FOR number = 1 TO 10 : READ flat$(number)
```

Cette instruction place les joueurs de golf dans leur appartement (flat$)



On se servira des variables de la manière habituelle, mais en faisant très attention aux indices.

Supposons maintenant que Tom et Sevvy veuillent échanger leurs appartements. En termes informatiques, l'un d'eux, par exemple Tom, utilisera un appartement provisoire (temp$) pour donner à Sevvy le temps de se déplacer :

On écrira :

```
LET temp$ = appartement$(1) : REMark Tom est dans l'appartement provisoire
LET flat$(1) = appartement$(3) : REMark On met Sevvy dans l'appartement de
Tom appartement$(1)
LET appartement$(3) = temp$ : REMark Tom occupe l'appartement de Sevvy
appartement$(3)
```

Le programme suivant place les noms des dix joueurs de golf dans un tableau nommé flat$ et écrit les noms des occupants, avec leurs indices, il échange les occupants des appartements 1 et 3, et donne à nouveau la liste pour montrer que l'échange a bien eu lieu.

```
100 REMark Les appartements des golfeurs
110 DIM appartement$(10,8)
120 FOR nombre = 1 TO 10 : READ appartement$(number)
130 printlist
140 exchange
150 printlist
160 REMark Fin du programme principal
170 DEFine PROCedure printlist
180 FOR nombre = 1 TO 10 : PRINT nombre, appartement$(nombre)
190 END DEFine
200 DEFine PROCedure exchange
210 LET temp$ = appartement$(1)
220 LET appartement$(1) = appartement$(3)
230 LET appartement$(3) = temp$
240 END DEFine
250 DATA "Tom","Graham","Sevvy","Jack","Lee"
260 DATA "Nick","Bernard","Ben","Greg","Hal"
```

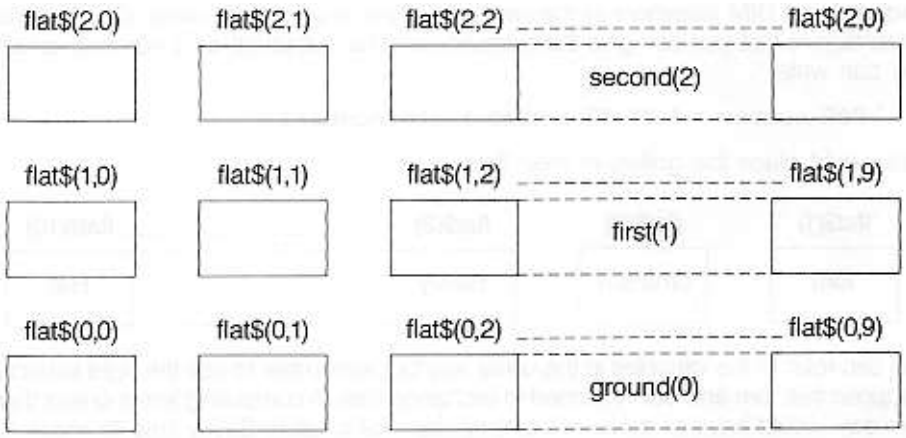
Sortie (ligne 130)	Sortie (ligne 150)
1 Tom	1 Sevvy
2 Graham	2 Graham
3 Sevvy	3 Tom
4 Jack	4 Jack
5 Lee	5 Lee
6 Nick	6 Nick
7 Bernard	7 Bernard
8 Ben	8 Ben
9 Gregg	9 Gregg
10 Hal	10 Hal

TABLEAUX A DEUX DIMENSIONS

Il est quelquefois plus pratique d'utiliser un tableau à deux dimensions comme trois niveaux de 10 appartements plutôt qu'une rangée de 30 appartements.

Supposons que 20 joueurs de golf, ou plus, aient besoin d'un appartement, alors qu'il existe un immeuble de trente appartements répartis sur trois niveaux de 10 appartements. Une méthode réaliste de représentation de l'immeuble est un tableau à deux dimensions.

On pourra alors représenter les trente variables de la manière suivante :



Si des instructions **DATA** nous fournissent 30 noms, une méthode convenable pour placer les noms sera :

```
120 FOR étage = 0 TO 2
130   FOR nombre = 0 TO 9
140     READ appartement$(étage,nombre)
150   END FOR nombre
160 END FOR étage
```

Il faut aussi une instruction **DIM** :

```
20 DIM flat$(2,9,8)
```

Le premier indice pourra varier de 0 à 2 (numéros d'étages) et le second de 0 à 9 (numéros de chambre). Le troisième paramètre indique la longueur maximale de chaque élément du tableau.

Nous ajoutons une séquence d'impression pour montrer que les joueurs sont dans leurs appartements, et nous remplaçons leurs noms par des lettres par économie de place :

```
100 REMark 30 Golfeurs
110 DIM appartement$(2,9,8)
120 FOR étage = 0 TO 2
130   FOR nombre = 0 TO 9
140     READ appartement$(étage,nombre) : REMark On entre les
golfeurs
150   END FOR nombre
160 END FOR étage
170 REMark Fin des entrées de données
180 FOR étage = 0 TO 2
190   PRINT "Etage numéro" ! Etage
200   FOR nombre = 0 TO 9
210     PRINT 'Appartement' ! nombre ! appartement$(étage,nombre)
220   END FOR nombre
230 END FOR étage
240 DATA "A","B","C","D","E","F","G","H","I","J"
250 DATA "K","L","M","N","O","P","Q","R","S","T"
260 DATA "U","V","W","X","Y","Z","@","£","$","%"
```

On aura en sortie :

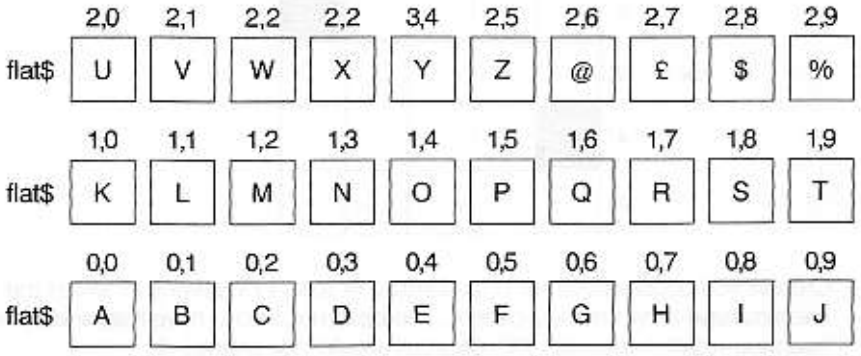
```
Etage numéro 0
Appartement 0 A
Appartement 1 B
Appartement 2 C
```

etc, jusqu'au trentième occupant.

DECOUPAGES DE TABLEAUX

La lecture de cette partie vous paraîtra peut-être difficile, bien qu'elle ait des analogies avec les sous-chaînes. Vous aurez certainement plus souvent besoin des sous-chaînes, que des découpages de tableaux, et vous pourrez à la rigueur sauter cette partie lors d'une première lecture.

Nous allons maintenant simplifier le problème des joueurs de golf dans leurs appartements afin d'illustrer le concept de découpage de tableaux. Les appartements sont numérotés de 1 à 9 et les noms sont exprimés par un seul caractère pour des raisons d'économie de place.



Les valeurs suivantes sont des découpages de tableaux, dont les valeurs sont déduites du tableau à deux dimensions précédent :

appartement$(1,3) représente un élément du tableau, contenant la lettre N

appartement$(1,1 TO 6) représente six éléments du tableau contenant les lettres L M N O P Q

Elément de tableau	Valeur
appartement$(1,1)	L
appartement$(1,2)	M
appartement$(1,3)	N
appartement$(1,4)	O
appartement$(1,5)	P
appartement$(1,6)	Q

appartement$(1) représente appartement$(1, 0 TO 9) soit dix éléments ayant pour valeurs K L M N O P Q R S T

On remarque sur ces exemples :

1. que l'on peut remplacer un indice par un intervalle de valeurs d'indices.
2. que si un indice est complètement absent, le système le remplace par la rangée complète (cas du 3ème exemple)

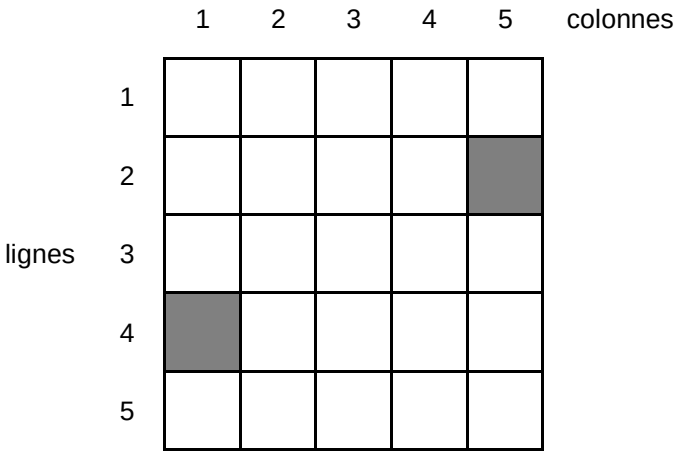
Les techniques de découpage de tableaux et de chaînes sont semblables, toutefois, on rencontre plus souvent les dernières.

PROBLEMES SUR LE CHAPITRE 13

1. TRI

   Lire dix nombres par une instruction DATA et les mettre dans un tableau. Rechercher dans le tableau le plus petit nombre et le mettre dans le premier élément d'un second tableau. Remplacer alors ce nombre du 1er tableau par un grand nombre. Même chose avec le plus petit nombre restant du premier tableau, et ainsi de suite jusqu'à ce que vous ayez obtenu un second tableau de nombres triés que vous afficherez.

2. MOTS CROISES



   Les mots croisés sont en général formés de grilles qui comportent un nombre impair de lignes et de colonnes dans laquelle les carrés noirs forment des figures symétriques par rapport à une droite. Cette symétrie est en fait une rotation de 180°.

   Remarquez que, après la rotation de 180 °, le carré ligne 4, colonne 1 devient le carré ligne 2, colonne 5.

   Ecrivez un programme qui génère et affiche une figure rotationnelle de cette sorte.

3. Modifier la figure du problème ci-dessus de sorte qu'il n'y ait pas de suites horizontale ou verticale de plus de quatre carrés blancs.

4. BATTAGE DE CARTES

   On identifie les cartes d'un jeu avec les nombres de 1 à 52 chargés dans un tableau. On peut facilement convertir ces nombres en valeurs de cartes. Les cartes sont réparties comme suit :

   - Choisir une position quelconque entre 1 et 51, par exemple 17.
   - Placer la carte correspondante (donc la 17) dans une case mémoire temporaire.
   - Ramener alors toutes les cartes qui étaient en position 52 à 18, en position 51 à 17.
   - Mettre alors la carte de la mémoire temporaire en position 52.
   - Faire la même chose avec les cartes en position de 1 à 50, de 1 à 49, etc. jusqu'à 1 à 2.
   - Ecrire le résultat.

5. Ecrire six instructions **DATA** contenant chacune un nom, une initiale et un numéro de téléphone. Choisir une structure de tableau convenable pour charger cette information et la lire dans le tableau.

Ecrire les données à l'aide d'une boucle **FOR** et expliquer pourquoi le format des donnés entrées, le format de stockage des données (tableaux), et le format de sortie ne sont pas nécessairement les mêmes.

CHAPITRE 14

STRUCTURE DE PROGRAMME

Dans ce chapitre nous allons reprendre l'essentiel de la structure d'un programme : boucles, décisions ou sélection. Nous avons essayé de présenter les choses aussi simplement que possible, mais SuperBASIC est en mesure de faire face à toutes les situations : de la plus simple à la plus complexe. Quelques passages de ce chapitre sont difficiles, et si vous êtes encore des programmeurs débutants, vous pouvez les sauter. Les parties importantes sont :

- les boucles
- les boucles imbriquées
- les décisions binaires
- les décisions multiples

La fin de cette partie « boucles » va devenir difficile, car nous allons montrer que SuperBASIC résoud certains problèmes tout simplement ignorés par d' autres langages.

BOUCLES

Dans cette partie nous illustrons le problème bien connu de répétitions, en simulant quelques scènes du Far West. L'histoire peut paraître imaginaire et triviale, mais elle offre une base simple de discussion et elle illustre les difficultés que l'on rencontre en programmation.

Exemple 1

Un bandit est abattu dans Old School House. Le sheriff a six balles dans son fusil. Simuler le tir des six coups.

Programme 1 :

```
100 REMark Western FOR
110 FOR balle = 1 TO 6
120   PRINT "Visez"
130   PRINT "Feu"
140 END FOR balle
```

Programme 2 :

```
100 REMark Western REPeat
110 LET balles = 6
120 REPeat bandit
130 PRINT "Visez"
140 PRINT "Feu"
150 LET balles = balles - 1
160 IF balles = 0 THEN EXIT bandit
170 END REPeat bandit
```

Ces deux programmes donnent la même sortie :

```
Visez
Feu
```

écrit six fois.

Si dans chaque programme, on remplace 6 par n'importe quel nombre, les deux programmes s'exécutent aussi correctement. Mais que se passe-t-il si le fusil est vide avant le tir des balles ?

Exemple 2

Supposons que quelqu'un ait secrètement enlevé les balles du fusil du sheriff. Que se passe-t-il si on remplace 6 par zéro ?

Programme 1 :

```
100 REMark Western FOR cas zéro
110 FOR balles = 1 to 0
120   PRINT "Visez"
130   PRINT "Feu"
140 END FOR balles
```

Il n'y a aucune sortie, ce qui est correct.

Programme 2 :

```
100 REMark Western REPeat Faux
110 LET balles = 0
120 REPeat bandit
130   PRINT "Visez"
140   PRINT "Feu"
150   LET balles = balles - 1
160   IF balles = 0 THEN EXIT bandit
170 END REPeat bandit
```

Ce programme contient deux erreurs :

1. Visez - Feu sont imprimés alors qu'il n'y a aucune balle
2. La variable « balles » passe à -1 en ligne 150 et est testée en ligne 160 : elle ne passera plus jamais à zéro, et le programme va boucler indéfiniment. On évitera cette boucle infinie en écrivant en ligne 160

   ```
   160 IF balles < 1 THEN EXIT bandit
   ```

Pour éviter l'autre faute, on devra placer correctement la condition **EXIT** avant les instructions **PRINT**.

Programme 3 :

```
100 REMark Western REPeat Cas zéro
110 LET balles = 0
120 REPeat Bandit
130   IF balles = 0 THEN EXIT Bandit
140   PRINT "Visez"
150   PRINT "Feu"
160   LET balles = balles - 1
170 END REPeat Bandit
```

Le programme est maintenant correct, quelle que soit la valeur initiale du nombre de balles, à condition qu'elle soit positive ou nulle. La méthode 2 correspond à la boucle **REPEAT** ... **UNTIL** de certains langages. La méthode 3 correspond à la boucle **WHILE** … **ENDWHILE** de certains langages. Cependant, la boucle **REPeat** … **END REPeat** avec **EXIT** est plus souple que l'une ou l'autre de ces deux boucles.

Si vous connaissez d'autres BASICs vous vous étonnez peut-être de n'avoir pas rencontré l'instruction **NEXT**. Nous allons bientôt l'introduire, mais nous allons d'abord montrer que les boucles **FOR** et **REPeat** ont la même structure et portent toutes deux un nom.

FOR nom =	(mot clé d'ouverture)	**REPeat** nom =
instructions	contenu	instructions
END FOR nom	mot clé de fermeture	**END REPeat** nom

De plus, la boucle **REPeat** doit comporter une instruction **EXIT** dans son contenu, sinon elle ne finira jamais. Remarquons aussi qu'après l'instruction **EXIT**, le contrôle revient à la ligne qui suit immédiatement la boucle. On peut employer l'instruction **NEXT** dans une boucle. Le contrôle reviendra alors à l'instruction qui succède au mot-clé **FOR** ou **REPeat**. On peut la considérer un

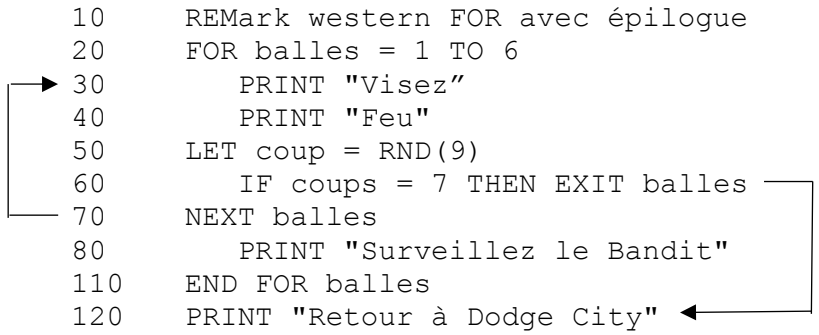
peu comme l'opposé de **EXIT**. L'une (**NEXT**) indique « recommencez », l'autre (**EXIT**) indique « C'est fini ».

Exemple 3

Nous sommes dans la même situation que dans l'exemple 1. Le sheriff a six balles dans son fusil et il s'apprête à tirer sur le bandit, mais deux conditions s'ajoutent.

1. S'il abat le bandit, il s'arrête de tirer et retourne à Dodge City.
2. S'il épuise ses balles avant d'avoir abattu le bandit, il demande à son assistant de surveiller le bandit, pendant que lui (le sheriff) retourne à Dodge City.

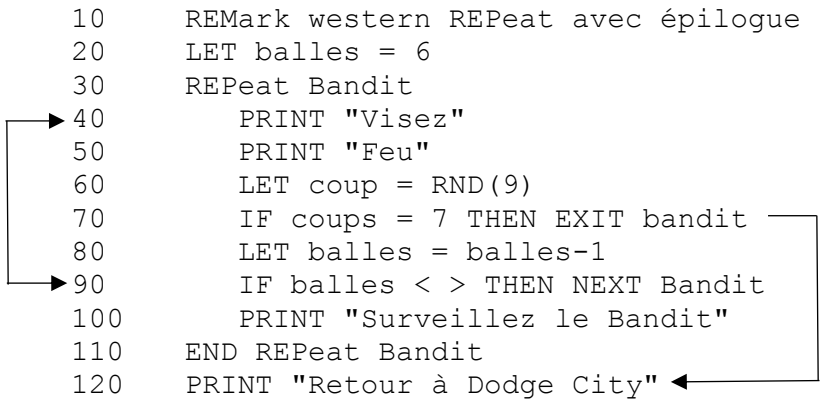
Programme 1 :



Dans cet exemple, le contenu entre **NEXT** et **END FOR** est une sorte d'épilogue qui ne sera exécuté que dans le cas où la boucle **FOR** s'exécute entièrement. S'il y a un **EXIT** prématuré, l'épilogue ne sera pas exécuté.

On pourrait obtenir la même chose avec une boucle **REPeat**, bien que cette méthode ne soit pas meilleure. Toutefois, elle vous aidera à comprendre des structures dont l'emploi est assez simple et la puissance suffisante pour faire face aux situations les plus enchevêtrées.

Programme 2



Le programme s'exécute correctement tant que le shériff a encore une balle à tirer. Il devient faux si on a en ligne 20 :

```
20   LET balles = 0
```

Vous pourriez trouver le shériff inconscient de tenter une telle entreprise sans avoir de balles du tout, et vous auriez raison. Mais ce qui nous importe maintenant, c'est la qualité de la structure. La solution à ce problème d'approvisionnement de coups, d'épuisement de balles avec épilogue, et du cas zéro, consiste en l'insertion d'une instruction supplémentaire :

```
35   IF balles = 0 THEN PRINT "Surveillez le bandit" : EXIT bandit
```

Avec une simple boucle on ne peut pas résoudre de problèmes plus complexes. Mais SuperBASIC en est capable si vous le désirez.

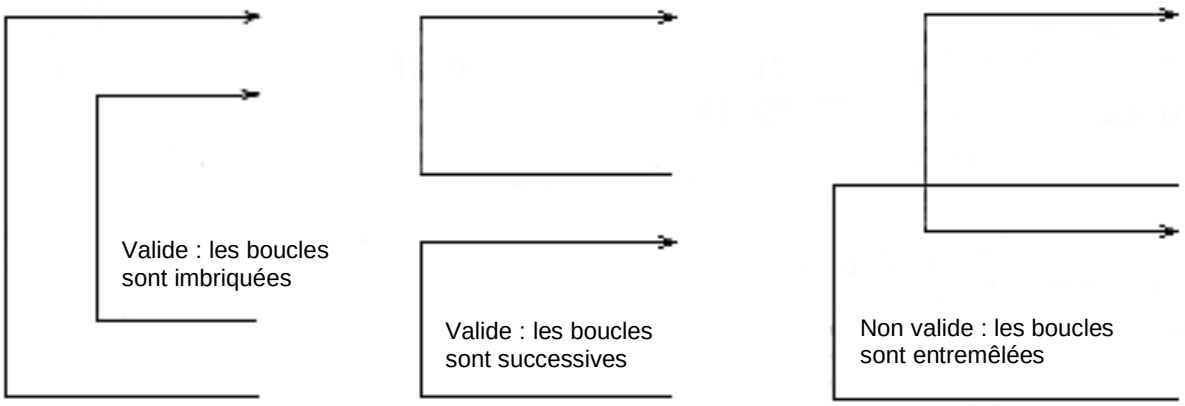
BOUCLES IMBRIQUEES

Considérons la boucle FOR suivante qui construit une rangée de points de couleurs aléatoires (sauf le noir).

```
10   REMark rangée de pixels
20   PAPER = 0 : CLS
30   LET ordonnée = 50
40   FOR abscisse = 20 TO 60
50      INK RND(2 TO 7)
60      POINT abscisse,ordonnée
70   END FOR abscisse
```

Ce programme construit une rangée de points de la manière suivante :

Si vous désirez construire par exemple 51 rangées de points il vous faudra une rangée pour chaque point dont l'ordonnée est comprise entre 50 et 80. Une autre boucle pourra vous permettre de construire ces 50 rangées, mais il faudra veiller à ce que l'une soit entièrement contenue dans l'autre, ou complètement en dehors. On représente souvent ces structures de boucles imbriquées de la manière suivante :



En SuperBASIC, cette règle s'applique à toutes les structures. Vous pourrez résoudre tous les problèmes en en tenant compte. Néanmoins, voici un exemple de boucles FOR imbriquées :

```
FOR ordonnées = 30 TO 80
   FOR abscisse = 20 TO 60
      POINT abscisse, ordonnée
   END FOR abscisse
END FOR ordonnée
```

En traduisant ceci dans un programme, nous avons le droit, non seulement d'attendre qu'il tourne, mais encore de savoir ce qu'il fait. Celui-ci construit un rectangle avec des rangées de pixels.

```
10   REMark rangées de pixels
20   PAPER 0 : CLS
30   FOR ordonnée = 30 TO 80
50      FOR abscisse = 20 TO 60
60         INK RND(2 TO 7)
70         POINT abscisse,ordonnée
80      END FOR abscisse
90   END FOR ordonnée
```

On peut imbriquer différentes structures. Remplaçons la boucle intérieure FOR du programme ci-dessus par une boucle REPeat. Nous sortirons de la boucle REPeat quand la couleur ayant le code 0 apparaîtra :

```
10    REMark REPeat in FOR
20    PAPER 0 :CLS
30    FOR ordonnée = 30 TO 80
40       LET abscisse = 19
50       REPeat points
60          LET couleur = RND(7)
70          INK couleur
80          LET abscisse = abscisse + 1
90          POINT abscisse,ordonnée
100         IF couleur = 0 THEN EXIT points
110      END REPeat points
120 END FOR ordonnée
```

Voici deux règles qu'il sera sage de respecter lorsque vous construirez vos programmes :

1. N'utilisez que les structures légitimes pour les boucles, et les décisions.
2. N'écrivez ces structures que l'une à la suite de l'autre ou l'une entièrement dans l'autre.

DECISIONS BINAIRES

On peut facilement illustrer les trois types de décisions binaires par l'exemple ci-dessous qui indique ce qu'il faut faire quand il pleut :

1er exemple :

```
10    REMark IF forme courte
20    LET pluie = RND (0 TO 1)
30    IF pluie THEN PRINT "ouvrir le parapluie"
```

2ème exemple :

```
10    REMark forme longue IF...  END IF
20    LET pluie = RND (0 TO 1)
30    IF pluie THEN
40       PRINT "prendre l'imperméable"
50       PRINT "ouvrir le parapluie"
60       PRINT "marcher vite"
70    END IF
```

3ème exemple :

```
10    REMark forme longue IF..ELSE..END IF
20    LET pluie = RND(0 TO 1)
30    IF pluie THEN
40       PRINT "prendre un bus"
50    ELSE
60       PRINT "marcher"
70    END IF
```

Dans tous ces exemples, il y a des décisions binaires. Les deux premiers exemples sont simples : ou bien il se passe quelque chose, ou non. Le troisième est une décision binaire plus générale, comportant deux possibilités distinctes, toutes deux définies : c'est une alternative.

Dans la forme longue, THEN est facultatif. Dans la forme courte, il peut être remplacé par "**:**"

Exemple :

Considérons un exemple plus complexe dans lequel il paraît naturel d'imbriquer des décisions binaires. Ce type d'imbrications peut porter à confusion et il ne faudra l'employer que dans les cas où il paraît le mieux approprié. Il est très important de disposer et d'imbriquer ces instructions avec le plus grand soin.

Analysons un texte et comptons le nombre de voyelles, de consonnes et autres caractères. Les espaces sont ignorés. Pour simplifier, le texte est tout en majuscules :

"L'HISTOIRE DES ORDINATEURS FUT ECRITE EN 1984"

Lire les données :

```
FOR chaque caractère
  IF lettre THEN
    IF voyelle
      incrémenter le nombre de voyelles
    ELSE
      incrémenter le nombre de consonnes
    END IF
  ELSE
     IF non espace THEN incrémenter le nombre des autres caractères
  END IF
END FOR
PRINT résultats
```

```
10    REMark compte de caractères
20    RESTORE : READ texte$
30    LET voyelle = 0 : cons = 0 ; autres = 0
40    FOR nombre = 1 TO LEN(texte$)
50      LET car$ = texte$(nombre)
60      IF car$>= "A" AND car$<="Z"
70        IF car$ INSTR "AEIOUY"
80          LET voyelles = voyelles + 1
90        ELSE
100         LET cons = cons+ 1
110       END IF
120     ELSE
130         IF car$<> " " THEN autres = autres + 1
140     END IF
150   END FOR nombre
160   PRINT "nombre de voyelles" ! voyelles
170   PRINT "nombre de consonnes" ! cons
180   PRINT "nombre d'autres caractères"! autres
190   DATA "L'HISTOIRE DES ORDINATEURS FUT ECRITE EN 1984"
```

Ce qui donnera :

```
Le nombre de voyelles est 15
le nombre de consonnes est 19
le nombre des autres est 5
```

DECISIONS MULTIPLES

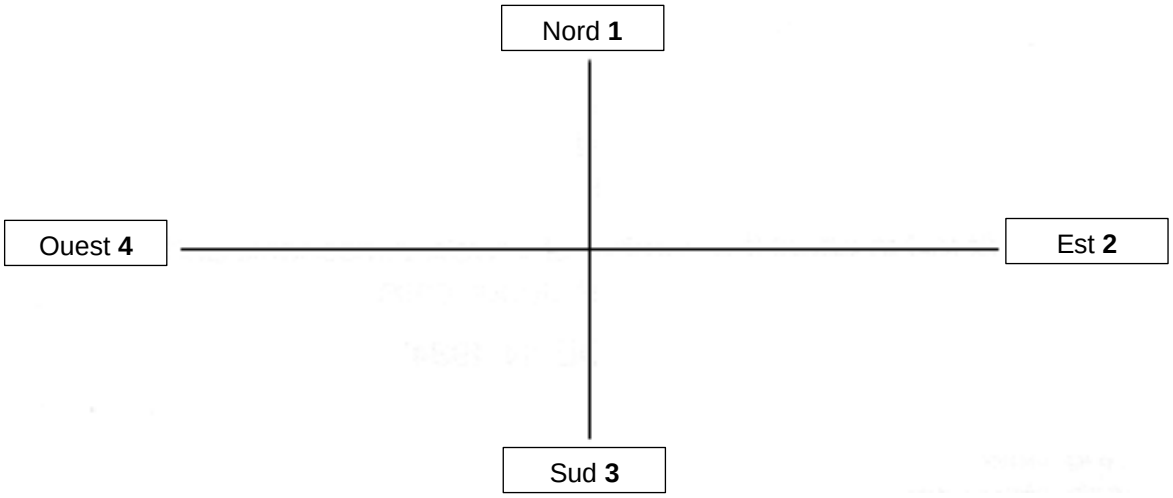
Lorsqu'il y a trois actions possibles, ou plus, et qu'aucune ne dépend d'un choix précédent, il est naturel d'utiliser l'instruction SELect qui permet de faire une sélection dans un nombre quelconque de possibilités.

Exemple :

Un serpent magique croît sans limite en ajoutant à chaque fois une partie de 20 unités de longueur maximale d'une couleur qui peut, soit être nouvelle, soit rester la même. Chaque nouvelle partie croît dans l'une des quatre directions Nord, Sud, Est, Ouest. Le serpent commence au centre de la fenêtre.

Méthode :

Tant que le serpent est à l'écran, vous choisissez une longueur et une couleur aléatoires. Vous sélectionnez une direction à l'aide d'un nombre 1, 2, 3 ou 4 comme sur la figure :



Principe :

Sélectionner le PAPIER
Construire le serpent au centre de la fenêtre
REPeat
  Choisir une direction, une couleur, une taille de croissance
  FOR unité = 1 jusqu'à croissance
      Faire croître le serpent vers le nord, le sud, l'est, l'ouest
  END FOR
END REPeat
PRINT message de fin

Programme :

```
10    REMark serpent magique
20    PAPER 0 : CLS
30    LET abscisse = 50 : ordonnée = 50
40    REPeat serpent
50      LET direction = RND(1 TO 4) :couleur = RND(2 TO 7)
60      LET croissance = RND (2 TO 20)
70      INK couleur
80      FOR unité = 1 TO croissance
90        SELect ON direction
100         ON direction = 1
```

```
110           LET ordonnée = ordonnée + 1
120         ON direction = 2
130           LET abscisse = abscisse + 1
140         ON direction = 3
150           LET ordonnée = ordonnée - 1
160         ON direction = 4
170           Let abscisse = abscisse - 1
180       END SELect
190       IF abscisse < 1 OR abscisse > 99 OR ordonnée < 1 OR
          ordonnée >99 THEN EXIT serpent
200       POINT abscisse,ordonnée
210     END FOR unité
220   END REPeat serpent
230   PRINT "Serpent hors de la bordure de l'écran"
```

La syntaxe de la structure SELect ON donne aussi la possibilité de faire une sélection dans une liste de valeurs comme suit :

```
5, 6, 8, 10 TO 13
```

On peut aussi utiliser cette structure pour commander une action qui ne sera exécutée que dans le cas où aucune des valeurs énoncées n'est trouvée. Voici la structure complète :

SELect ON nombres
ON nombres = liste de valeurs
instructions
ON nombres = liste de valeurs
instructions
etc...
…
…
…
ON nombres = REMAINDER
instructions
END SELect

où 'nombres' est une variable numérique quelconque et la clause REMAINDER (qui permet d'exprimer les cas restants) est optionnelle.

Il y a aussi une forme courte de la structure SELect ON. Par exemple :

```
10    INPUT nombre
20    SELect ON nombre = 0 TO 9 :PRINT "chiffre"
```

PROBLEMES SUR LE CHAPITRE 14

1. Charger 10 nombres dans un tableau, et faire un tri de la manière suivante : comparer les deux premiers nombres et les échanger si le premier n'est pas le plus petit, même chose avec les deux nombres suivants (le deuxième et le troisième) et ainsi de suite jusqu'au neuvième et dixième. La première exécution des neuf comparaisons et échanges possibles nous assure que le plus grand nombre doit avoir atteint la dernière position. La neuvième nous assure que le plus petit nombre a gagné la première position. Pour 10 nombres, il faut donc neuf exécutions de neuf comparaisons.

2. Trouver une méthode plus rapide pour faire ce tri.

3. Un commissaire-priseur souhaite vendre une vieille pendule et il a reçu l'instruction de faire une première offre à 50 F. S'il n'y a aucune enchère, il descendra à 40 F., puis 30 F., puis 20 F., mais pas plus bas. S'il n'y a pas d'enchère, l'horloge sera retirée de la vente. Quand les enchères démarrent, il augmente de 5 F. à chaque fois, jusqu'à l'enchère finale. Si l'enchère finale est 35 F. ou plus, l'horloge est vendue, sinon elle est retirée de la vente.

   On vous demande de simuler la vente, en utilisant un dé à six faces, la sortie d'un six étant l'équivalent du démarrage d'une enchère. Comptez ensuite quatre enchères possibles.

4. Nous sommes encore au Far-West. Le shérif, sans munitions, souhaite arrêter le brigand caché dans la forêt. Il chevauche parmi les arbres en incitant le brigand à tirer. Il espère pouvoir surprendre le brigand après qu'il ait tiré six coups, et qu'il rechargera son fusil. Simulez la rencontre et donnez au brigand une chance sur vingt d'abattre le shérif à chaque coup. Si le shérif n'est pas abattu au dixième coup, il arrête le brigand.

5. Les instructions du shérif à son adjoint seront les suivantes :
   « Si le fusil est vide, recharge-le, et sinon, tire jusqu'à ce que tu touches le bandit ou jusqu'à ce qu'il se rende. Si Mexico Pete réplique, pars vite."

   Ecrivez un programme qui satisfait à toutes ces situations :

   Quoi qu'il advienne, reviens à Dodge City.
   Si Mexico Pete réplique, reviens immédiatement.
   Si le pistolet est vide, recharge-le.
   Si le pistolet n'est pas vide, demande au bandit de se rendre.
   Si le bandit se rend, arrête-le.
   S'il ne se rend pas, tire une fois.
   Si le bandit est touché, arrête-le, et soigne ses blessures.

   Accordez une quantité illimitée de munitions. Utilisez un dé à vingt faces fictives, où sept veut dire "se rendre" et treize veut dire "le bandit est touché".

CHAPITRE 15

PROCEDURES ET FONCTIONS

Dans la première partie de ce chapitre, nous expliquerons les caractéristiques essentielles des procédures et fonctions de SuperBASIC. Pour ce faire, nous utilisons de très simples exemples qui, une fois compris, pourront être appliqués dans les situations les plus complexes.

Après cette première partie, un autre paragraphe vous expose « pourquoi des procédures ». Si vous assimilez ce passage, vous serez capable d'utiliser les procédures et les fonctions avec un succès qui ira croissant.

SuperBASIC met d'abord à votre disposition des choses simples, puis vous offre plus si vous le désirez. Des facilités supplémentaires et quelques points techniques sont exposés dans la seconde partie de ce chapitre, mais vous pouvez les omettre, tout au moins à la première lecture ; vous vous trouverez malgré tout en meilleure position que la plupart des utilisateurs des anciens types de BASIC.

VALEUR DES PARAMETRES

Vous avez vu, dans les précédents chapitres, comment passer une valeur dans une procédure. Voici un autre exemple :

Exemple

Au restaurant "Chan% Chinese", il n'y a que six plats au menu :

Plat principal	Dessert
1 Crevettes	4 Glace
2 Poulet	5 Beignets
3 Assiette garnie	6 Lychees

Chan a une méthode très simple pour calculer ses prix. Il calcule en pence, de la manière suivante :
Pour un plat principal : 300 + 10 fois le numéro du menu.
Pour un dessert : 12 fois le numéro du menu.

Ainsi un client qui commande une assiette garnie et une glace paiera :
300 + 10*3 + 12*4 = 378 pence.

Une procédure, nommée item, a comme paramètre le numéro de menu et calcule le prix :

```
10   REMark coût d'un plat
20   item 3
30   item 4
40   DEFine PROCedure item(num)
50         IF num<=3 THEN LET prix=300 + 10*num
60         IF num>=4 THEN LET prix = 12*num
70         PRINT ! Prix !
80   END DEFine
```

En sortie vous aurez : 330 48

Le programme principal utilise les paramètres 3 et 4. La procédure est définie avec un paramètre formel, num, qui prend la valeur que lui passe le programme principal. Notez que les paramètres formels doivent être entre parenthèses, alors que les paramètres actuels n'en ont pas besoin.

Exemple

Supposons maintenant que la variable utilisée soit aussi utilisée dans le programme principal, mais pour représenter quelque chose d'autre, par exemple le prix d'un verre de bière, disons 70 p. Le programme suivant ne donne pas le bon résultat :

```
10   REMark prix global
20   LET prix = 70
30   item 3
40   item 4
50   PRINT ! prix !
60   DEFine PROCedure item(num)
70        IF num <= 3 THEN prix = 300 + 10 * num
80        IF num >= 4 THEN prix = 12 * num
90   PRINT ! prix !
100  END DEFine
```

Vous pourrez avoir en sortie : 330 48 48

Le prix de la bière a été modifié par la procédure. Nous dirons que la variable prix est globale, parce qu'elle peut être utilisée n'importe où dans la procédure.

Exemple

Rendre la variable prix locale à la procédure, cela signifie que SuperBASIC la considérera comme une variable qui ne pourra être accédée qu'à partir de la procédure. La variable prix du programme principal sera différente, même si elle a le même nom.

```
10   REMark LOCAL prix
20   LET prix = 70
30   item 3
40   item 4
50   PRINT ! prix
60   DEFine PROCedure item(num)
70        LOCAL prix
80        IF num <= 3 THEN LET prix = 300 + 10*num
90        IF num >= 4 THEN LET prix = 12*num
100  PRINT ! prix
110  END DEFine
```

sortie : 330 48 70

Cette fois-ci tout se passe correctement. La ligne 70 rend la variable prix locale à la procédure, donc n'appartenant qu'à elle. L'autre variable prix n'est pas affectée. Vous pouvez remarquer que les variables locales sont quelque chose de très pratique.

Exemple

Les variables locales sont si pratiques que l'on écrit souvent des procédures locales avec des paramètres formels. Par exemple, et bien que nous ne l'ayons pas mentionné avant, des paramètres tels que "num" dans les programmes ci-dessus ne peut se confondre avec la variable du programme principal. Montrons-le en enlevant l'instruction LOCAL du programme précédent et en appelant "num" le prix de la bière. Num étant local à la procédure, le programme tourne correctement :

```
10    REMark LOCAL paramètre
20          LET num = 70
30          item.3
40          item 4
50          PRINT num
60          DEFine PROCedure item(num)
70                IF num <= 3 THEN LET prix = 300 + 10*num
80                IF num >= 4 THEN LET prix = 12*num
90                PRINT ! prix
100         END DEFine
```

sortie = 330 48 70

PARAMETRES VARIABLES

Jusqu'à présent nous n'avons utilisé les paramètres de procédures que pour passer des valeurs dans la procédure. Mais supposons maintenant que le programme principal ait besoin du prix du plat principal et du dessert pour calculer le montant de la facture. On fera cela aisément à l'aide d'un autre paramètre dans la procédure appelée. Celui-ci devra être une variable car il reçoit une valeur de la procédure. C'est pourquoi nous l'appelons un paramètre variable et il doit être associé à un autre paramètre variable dans la définition de la procédure.

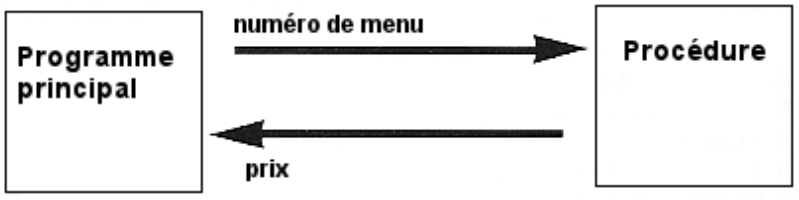
Exemple

Utilisons les paramètres variables cout_1 et cout_2 pour recevoir les valeurs des prix de la procédure. Demandons au programme principal de calculer le prix total.

```
10    REMark paramètre variable
20          LET num = 70
30          item 3, cout_1
40          item 4, cout_2
50          LET note = num + cout_1 + cout_2
60          PRINT note
70    DEFine PROCedure item(num,prix)
80          IF num <= 3 THEN LET prix = 300 + 10*num
90          IF num >= 4 THEN LET prix = 12*num
100   END DEFine
```

sortie 448

Les paramètres num et prix sont tous deux locaux de façon automatique, il n'y a donc ainsi aucun problème. Le diagramme ci-dessous montre le transfert de l'information du programme principal vers la procédure et inversement.



Pour l'instant arrêtons là notre étude des procédures.

FONCTIONS

Vous savez déjà comment procède une fonction du système. Par exemple, la fonction :

```
SQRT(9)
```

calcule la valeur 3, racine carrée de 9. Nous dirons que la fonction donne la valeur 3. Une fonction peut comme une procédure, avoir un ou plusieurs paramètres, mais la particularité d'une fonction est de ne donner qu'une seule valeur. Cela implique que l'on peut l'utiliser dans des expressions déjà existantes. Vous pouvez écrire :

```
PRINT 2*SQRT(9)
```

et vous obtiendrez 6. Ainsi une fonction se comporte comme une procédure à un ou plusieurs paramètres, et un paramètre unique pour le résultat, qui est le nom de la fonction.

Les paramètres peuvent être non numériques :

```
LEN("chaîne")
```

a pour argument une chaîne, mais donne la valeur numérique 6.

Exemple

Réécrivons le dernier programme en utilisant la variable paramètre "prix" comme une fonction.

```
10 REMark fonctions
20 LET num=70
30 LET note=num+prix(3)+prix(4)
40 PRINT note
50 DEFine FuNction prix(num)
60    IF num<=3 THEN cout = 300+10*num
70    IF num>=4 THEN cout = 12*num
80 RETurn cout
90 END DEFine
```

résultat 448

Notez la simplicité de l'appel de la fonction. Il y a une autre façon de définir la valeur de la fonction ; il utilise le mot RETURN comme suit

```
10    REMark Fonction avec RETURN
20    LET num = 70
30    LET note = num + prix(3) + prix(4)
40    PRINT note
50    DEFine FuNction prix(num)
60    IF num <= 3 THEN RETURN 300 + 10*num
70    IF num >= 4 then RETURN 12*num
80    END DEFine
```

résultat 448

Par souci de compatibilité avec d'autres BASICs, il a été défini une forme courte de définition de fonction.

Exemple

Utilisons la forme courte pour définir la fonction "diff" qui calcule la différence des longueurs de deux chaînes.

```
10    REMark fonction condensée
20    PRINT diff("chaine longue","courte")
30    DEFineFuNction diff(long,court) = LEN(long)-LEN(court)
```

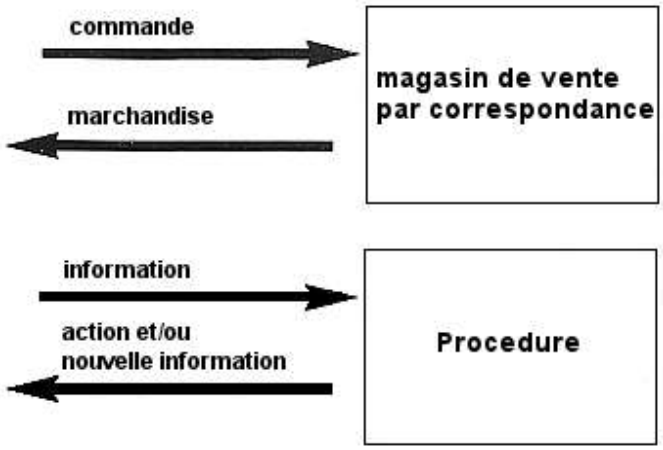
Résultat 7

L'abrévation de FuNction est FN pour être compatible avec d'autres BASICs.

POURQUOI DES PROCEDURES ?

Le concept final d'une procédure peut être l'établissement d'une comparaison entre une procédure et une « boite noire » qui reçoit des informations spécifiques de l'extérieur et exécute un certain nombre d'opérations, dont certaines peuvent renvoyer des informations à l'extérieur ; l'extérieur pouvant être le programme principal ou une autre procédure.

Le terme « boite noire » est employé pour montrer que le travail qui se déroule à l'intérieur n'est pas très important, mais qu'il faut l'insérer à ce qui y entre, et à ce qui en sort. Par exemple, si une procédure utilisant la variable « compte » change la valeur de cette variable, cela pourrait affecter une variable de même nom dans le programme principal. Pensons à un magasin de vente par correspondance. Vous passez une commande et vous payez ; vous recevez des marchandises. Une information est envoyée par une procédure et vous avez en retour une action et/ou une autre information.



Vous ne voulez pas que le magasin de vente par correspondance utilise votre nom, votre adresse ou autres informations pour autre chose. De la même façon, vous ne voulez pas qu'une procédure modifie les valeurs des variables du programme principal.

Naturellement, on pourrait s'assurer de ne pas utiliser le même nom de variable deux fois. Mais nous avons montré dans ce chapitre comment éviter les erreurs même si on a oublié les noms des variables utilisées dans une procédure particulière.

Le second but, dans l'emploi des procédures est d'obtenir un programme modulaire. Au lieu d'avoir un long programme principal, il vaut mieux partager le travail en parties assez petites pour qu'elles soient faciles à comprendre et à contrôler. Ces parties sont des procédures, reliées les unes aux autres par des appels de procédures, dans un ordre déterminé.

Un troisième but est d'éviter d'écrire plusieurs fois les mêmes lignes. Il est tout-à-fait possible d'écrire une procédure une fois et de l'appeler plusieurs fois si nécessaire.

Voici un autre exemple qui met en valeur la modularité de la méthode procédurale.

Exemple

On a commandé six plats chez Chan's, où le menu est :

Numéro	Plat	Prix
1	Crevettes	3.50
2	Poulet	2.80
3	Assiette garnie	3.30

Ecrire des procédures pour les tâches suivantes :

1. Construire deux tableaux contenant le menu, les plats et les prix.
   Utiliser l'instruction DATA.
2. Simuler une commande de six plats choisis de façon aléatoire, par une procédure « choisir » et compter combien de fois chaque plat est choisi.
3. Ecrire une procédure « garçon » qui prend ces trois nombres, envoie le prix de la commande dans le programme principal, en utilisant le paramètre « cout ». La procédure « garçon » appelle deux autres procédures « calcul » et « cuisson », qui calcule le prix et simule le travail en cuisine.
4. La procédure « cuisson » imprime simplement le numéro requis et le nom de chaque plat.

Le programme principal appelle les procédures selon leur nécessité, obtient le prix total à partir d'une procédure « garçon », ajoute 10% pour le service, et écrit le montant total de la note.

Schéma

Ce programme illustre les passages de paramètres de niveau assez complexe, et nous l'expliquons pas à pas avant de l'écrire.

```
10   REMark Procédures
20   DIM item$(3,15) ,prix(3), plat(3)
30   LET service = 0.9 : RESTORE
40 carte
100 DEFine PROCedure carte
110  FOR k = 1 TO 3
120  READ item$(k)
130  READ prix(k)
140  END FOR k
150 END DEFine
370 DATA "Crevettes",3.5,"Poulet",2.8,"Assiette garnie", 3.3
```

Le nom des plats et leurs prix sont dans les tableaux 'item$' et 'prix'.

L'étape suivante est de choisir un numéro de même pour chacun des six clients. On garde dans le tableau 'plat' le compte des numéros de chaque plat.

```
50  choix plat
160 DEFine PROCedure choisir (plat)
170 FOR choix = 1 TO 6
180 LET nombre = RND(1 TO 3)
190 LET plat(nombre) = plat(nombre) + 1
200 END FOR choix
```

Notez que le paramètre formel 'plat' est à la fois
- local pour la procédure choix
- un tableau du programme principal

Les trois valeurs sont passées dans le tableau global aussi appelé 'plat'. Puis ces valeurs sont passées dans la procédure 'garçon'.

```
60  garçon plat, note
220 DEFine PROCedure garçon(plat,cout)
230  addition plat,cout
240  cuisson plat
250 END DEFine
```

Le garçon fait passer le nombre de plats commandés de chaque sorte à la procédure « addition » qui calcule le prix et le renvoie.

```
260 DEFine addition (plat,total)
270  LET total = 0
280  FOR k = 1 TO 3
290    LET total = total + plat(k)*prix(k)
300  END FOR k
310 END DEFine
```

Le garçon passe alors l'information au cuisinier qui écrit simplement le nombre de plats commandés de chaque sorte.

```
320 DEFine PROCedure cuisson(plat)
330  FOR c = 1 TO 3
340  PRINT ! plat(c) item(c)
350  END FOR c
360 END DEFine
```

Encore une fois le tableau 'plat' est local dans la procédure 'cuisson'. Il reçoit l'information utilisée par la procédure par son instruction PRINT.

Programme complet

```
10   REMark Procedures
20   DIM item$(315),prix(3), plat(3)
30   REMark*** PROGRAMME ***
40   LET service = 0.9 : RESTORE : CLS
50   carte
60   choisir plat
70   garçon plat, note
80   LET note = note + service
90   PRINT "cout total :F" ! note
100  REMark *** PROCEDURE DEFINITIONS ***
110  DEFine PROCedure carte
120    FOR k = 1 TO 3
130     READ item$(k)
140     READ prix(k)
150    END FOR k
160  END DEFine
170  DEFine Procedure choisir plat}
180    FOR choix = 1 TO 6
190      LET nombre = RND(1 TO 3)
200      LET plat(nombre) = plat(nombre) + 1
210    END FOR choix
220  END DEFine
230  DEFine PROCedure garçon(plat,cout)
240   addition plat,cout
250    cuisson plat
260  END DEFine
270  DEFine PROCedure addition (plat,total)
280    LET total = 0
290    FOR k = 1 TO 3
300      LET total = total + plat(k)*prix(k)
310    END FOR k
320  END DEFine
330  DEFine PROCedure cuisson (plat)
340    FOR c = 1 TO 3
350      PRINT ! plat(c) ! item$(c)
380    END FOR c
```

```
370   END DEFine
380   REMark *** DONNEES DU PROGRAMME ***
390   DATA "Crevettes",3.51"Poulet",2.8,"Assiette garnie",3.3
```

La sortie dépend du choix aléatoire des plats, mais nous donnons un exemple de sortie possible :

3 crevettes
1 poulet
2 assiettes garnies
Prix total = 20.40 F.

Commentaire

Evidemment, l'emploi de procédures et de paramètres dans un tel programme simple n'est pas nécessaire, mais imaginez que chaque sous-tâche soit plus complexe.Alors l'emploi de procédures permettrait une structure modulaire du programme, avec test à chaque niveau. L'exemple ci-dessus illustre bien les principales relations et notations des procédures.

De la même façon l'exemple suivant illustre l'emploi des fonctions.

Remarquons que dans le programme précédent, les procédures « garçon » et « addition » donnent toutes deux une valeur unique. Réécrivons ces procédures comme des fonctions en montrant les modifications qui découlent de ce changement.

```
DEFine FuNction garçon(plat)
   cuisson plat
   RETURN addition plat
END DEFine
DEFine FuNction addition(plat)
   LET total = 0
   FOR k = I TO 3
      LET total = total + plat*prix(k)
   END FOR k
END DEFine
```

On peut aussi appeler la fonction « garçon » d'une autre manière :

```
LET note = garçon(plat)
```

Ce programme fonctionne comme précédemment ; il faut noter une économie de paramètres bien que la structure du programme soit la même. Ceci s'explique par le fait que les noms de fonctions servent aussi de paramètres.

Exemple

Toutes les variables utilisées comme paramètres formels dans les procédures ou fonctions sont « sauvées » parce qu'elles sont locales. Quelles variables utilisées dans les procédures et fonctions ne le sont-elles pas ? Quelle instruction supplémentaire faut-il pour les rendre locales ?

Modification de programme

Les variables 'k', 'choix' et 'num' ne sont pas locales. Les modifications à faire sont donc :

```
LOCAL k
LOCAL choix,num
```

PARAMETRES SANS TYPE

Les paramètres formels ne doivent pas avoir de type. Nous n'avions pas encore mentionné cela, parce qu'il est tout à fait possible de travailler correctement sans le savoir. Ces paramètres auront la même définition, qu'ils contiennent des valeurs numériques ou non, il faudra les déclarer sans spécifier leur type. Le programme suivant va vous en persuader :

```
10 REMark nombre ou mot
20 garçon 2
30 garçon "poulets"
40 DEFine PROCedure garçon (item)
50   PRINT item
60 END DEFine
```

Sortie : 2 poulets

C'est seulement au moment de l'appel de la procédure et de l'arrivée du paramètre que son type est déterminé.

DOMAINE DES VARIABLES

Dans le programme suivant, cherchons lequel des deux nombres sera donné en sortie :

```
10 REMark scope
20 LET nombre = 1
30 test
40 DEFine PROCedure test
50   local nombre
60   LET nombre = 2
70   PRINT nombre
80 essai
90 END DEFine
100 DEFine PROCedure essai
110  PRINT nombre
120 END DEFine
```

Il est bien évident que le premier nombre écrit sera 2, mais la variable nombre de la ligne 110 est-elle globale ? La réponse est que la valeur de « nombre » dans la ligne 60 sera transportée dans la procédure « essai ». Si une variable est locale dans une procédure, elle conserve sa valeur dans une seconde procédure appelée par la première.

De même si la procédure « essai » est appelée par le programme principal, la variable « nombre » aura la même valeur dans le programme principal et la procédure « essai ». Les implications peuvent paraître bizarres à première vue, mais elles sont logiques :

1. La variable « nombre » de la ligne 20 est globale
2. La variable « nombre » de la procédure « essai » est nettement locale à la procédure
3. La variable « nombre » de la procédure « essai » appartient à la partie du programme qui l'appelle en dernier.

Nous vous avons donné une vue d'ensemble de beaucoup de concepts dans ce chapitre, parce que SuperBASIC est très puissant au niveau des fonctions et procédures.

Toutefois il ne faut pas vous attendre à les employer tous dans l'immédiat. Utilisez-les de manière simple dans un premier temps. Elles vous seront malgré tout très utiles.

PROBLEMES SUR LE CHAPITRE 15

1. Six employés sont identifiés uniquement par leur nom de famille. Chaque employé a des cotisations de retraite exprimées par un pourcentage. Les données suivantes représentent les salaires et les cotisations de retraite des six employés.

Benson	13,800	6.25
Hanson	8,700	6.00
Johnson	10,300	6.25
Robson	15,000	7.00
Thomson	6,200	6.00
Watson	5,100	5.75

Ecrire des procédures pour :

- entrer les données dans des tableaux
- calculer le montant actuel des cotisations de retraite
- sortir les listes des noms et cotisations

Relier les procédures à un programme principal qui les appelle.

2. Ecrire une fonction « select » avec deux arguments « intervalle » et « erreur ». La fonction calcule un nombre aléatoire dans l'intervalle donné, dont la valeur doit être différente de « erreur ».

   Utiliser cette fonction dans un programme qui choisit une couleur de fond aléatoire puis trace des cercles aléatoires de couleur aléatoire, de sorte qu'aucun ne soit de la même couleur que le fond.

3. Réécrire la solution de l'exercice n°1 de manière qu'une fonction « pension » prenne les salaires et les taux des cotisations de retraite comme arguments, et donne le montant des cotisations de retraite.

   Utilisez deux procédures, une qui entre les données, et une qui donne les résultats à l'aide de la fonction « pension ».

4. Ecrire :
   - une procédure qui construit un paquet de cartes une procédure qui distribue les cartes (de façon aléatoire)
   - une fonction qui prend comme argument un nombre et donne une chaîne décrivant la carte.
   - une procédure qui donne et affiche quatre mains de poker de cinq cartes chacune.
   - un programme principal qui appelle les procédures ci-dessus (voir chapitre 16 la présentation d'un tel problème).

CHAPITRE 16

QUELQUES TECHN IQUES

Dans ce dernier chapitre, nous présentons quelques applications de concepts et facilités déjà exposés et nous montrons quelques aspects des développements possibles.

SIMULATION D'UN JEU DE CARTES

Il est facile de changer et de manipuler les cartes d'un jeu en les représentant par les nombres 1 à 52. Voici comment transformer un tel nombre pour obtenir la carte correspondante. Prenons par exemple le nombre 29. On peut décider que :

- les cartes de 1 à 13 sont les cœurs
- les cartes de 14 à 26 sont les trèfles
- les cartes de 27 à 39 sont les carreaux
- les cartes de 40 à 52 sont les piques.

La carte 29 est donc un carreau. On peut programmer le QL pour qu'il fasse ce travail :

```
LET couleur = (carte-1) DIV 13
```

Ceci donnera une valeur comprise entre 0 et 3 qui permettra de reconnaître la couleur de la carte.

On pourrait aussi écrire :

```
LET valeur = carte MOD 13
LET valeur = 0 THEN LET valeur = 13
```

pour connaître la valeur de la carte 29 dans sa couleur qui est ici carreau.

Programme

On peut écrire les nombres de 1 à 13 de la manière suivante : As,2,3... valet, dame, roi, ou encore mieux « deux de cœur ». Le programme suivant affiche le nom de la carte qui correspond au nombre entier.

```
100 REMark carte
110 DIM suitnom$(4,8),carteval$(13,5)
120 CLS:LET d$="de"
130 constitution
140 REPeat cartes
150   INPUT "entrer un numero de cartes de 1 a 52:"!carte
160   IF carte < 1 OR carte > 52 THEN EXIT cartes
170   LET couLeur=(carte-1) DIV 13+1
180   LET valeur=carte MOD 13
190   IF valeur=0 THEN LET valeur=13
200   PRINT carteval$(valeur)!d$!suitnom$(couleur)
210 END REPeat carte
220 DEFine PROCedure constitution
230   RESTORE
240   FOR s=1 TO 4:READ suitnom$(s)
250   FOR v=1 TO 13:READ carteval$(v)
260 END DEFine
270 DATA 'coeur',trefle,'carreaux','pique'
280 DATA 'as', 'deux', 'trois', 'qutre', 'cinq', 'six', 'sept'
290 DATA 'huit', 'neuf', 'dix', 'valet', 'dame', 'roi'
```

Entrée et sortie :

13
roi de cœur
49
dix de pique
27
as de carreau

Commentaire

Notez l'emploi des instructions DATA, ce qui constitue un fichier permanent de données que le programme utilisera toujours. Les données qui changent de valeur à chaque exécution de programme sont entrées par une instruction INPUT. Si la valeur de la donnée entrée est connue avant l'exécution du programme, il est tout à fait correct d'utiliser une autre instruction READ et d'autres instructions DATA. Cette méthode permet un meilleur contrôle.

FICHIER DE CARACTERES

Le programme suivant construit un fichier de cent nombres :

Fichier numérique

```
10 REMark fichier nombres
20 OPEN NEW #6, MDV1 nombres
30 FOR num = 1 TO 6
40   PRINT#6,num
50 END FOR num
60 CLOSE#6
```

Après l'exécution du programme, vérifiez que le nom du fichier « nombres » est bien dans le catalogue en tapant :

```
DIR MDVI nombres
```

Vous pouvez visualiser le fichier sans aucun formatage en faisant une copie du microdrive vers l'écran :

```
COPY MDVI nombres to SCR_
```

Vous pouvez aussi utiliser le programme suivant pour lire le fichier et afficher ses enregistrements à l'écran :

```
10 REMark fichier lu
20 OPEN IN #6,MDV1_nombres
30 FOR num = 1 TO 100
40   INPUT #6, item
50   PRINT ! item !
60 END FOR num
70 CLOSE #6
```

Il est possible de modifier ce programme pour avoir une sortie présentée autrement.

Fichier de caractères

De la même façon, les programmes suivants construisent un fichier de lettres choisies aléatoirement et lisent le fichier.

```
10 REMark fichier lettres
20 OPEN NEW #6,MDV1_Letfich
30 FOR num = 1 TO 100
40   LET car$ = CHR$(RND (65 TO 90))
50   PRINT #6,car$
60 END FOR num
70 CLOSE #6
```

```
10 REMark obtention de lettres
20 OPEN_IN #6,MDV1_Letfich
30 FOR num = 1 TO TOO
40   INPUT #6,item$
50   PRINT ! item$
60 END FOR num
70 CLOSE #6
```

CREATION D'UN FICHIER

On se propose de créer un fichier tout simple, contenant des noms et des numéros de téléphone.

RON	678462
GEOFF	896487
ZOE	249386
BEN	584621
MEG	482349
CATH	438975
WENDY	982387

Voici le programme :

```
10 REMark numeros de téléphone
20 OPEN NEW #6,mdvl_tel
30 FOR enreg=1 TO 7
40 INPUT 'NOM   :';nom$
45 INPUT 'TEL   :';num$
50 PRINT #6,nom$,num$
60 END FOR enreg
70 CLOSE #6
```

Tapez RUN, puis ENTREE, un nom suivi de ENTREE, puis un autre, etc... jusqu'à sept fois.

Remarquez que les données sont stockées dans une zone tampon (buffer), jusqu'à ce que le système soit prêt à transférer le lot dans le Microdrive. Il y aura donc un seul accès au microdrive.

COPIE DE FICHIER

Dès qu'un fichier a été créé, il est possible de le copier. Voici la commande :

```
COPY MDV1_TEL to MDV2_TEL
```

LECTURE DE FICHIER

Pour être certain qu'un fichier existe, et que sa forme est correcte, il est possible de le lire à partir du microdrive en l'affichant à l'écran. Voici la commande :

```
COPY MDV2_TEL to SCR
```

A la sortie de l'écran, l'espace entre le nom et le numéro de téléphone ne sera pas généré, mais il y aura un saut à la ligne après chaque enregistrement. La sortie sera :

RON678462
GEOFF896487
ZOE249386
BEN584621
MEG482349
CATH438975
WENDY982387

Le programme suivant permet de contrôler la présentation des données en sortie :

```
10 REMark lecture des n° de téléphone
20 OPEN_IN#5,MDV1_tel
30 FOR enreg = 1 TO 7
40   INPUT#5,E$
50   PRINT, E$
60 END FOR enreg
70 CLOSE #5
```

Les données sont affichées, comme ci-dessus, mais cette fois, chaque paire de champs est contenue dans la variable « E » avant qu'elle soit écrite à l'écran. Vous avez donc la possibilité de la manipuler si vous le désirez.

LE TRI A INSERTION

Voici les couleurs disponibles en mode écran basse résolution (numérotées de 0 à 7)

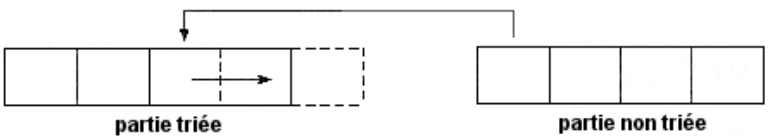
noir, bleu, rouge, magenta, vert, cyan, jaune, blanc
black, blue, red, magenta, green, cyan, yellow, white

Exemple

Ecrire un programme qui tire les couleurs et les classe en ordre alphabétique (en anglais).

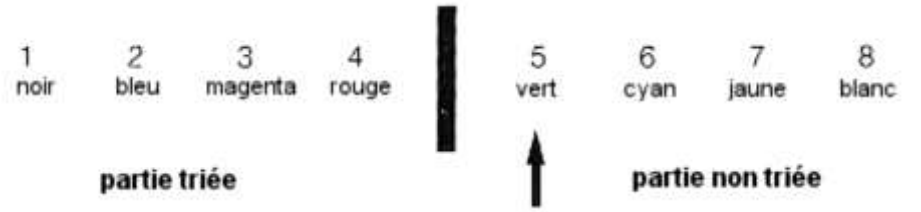
Méthode

Nous plaçons les huit couleurs dans un tableau couleur$ que nous divisons en deux parties :

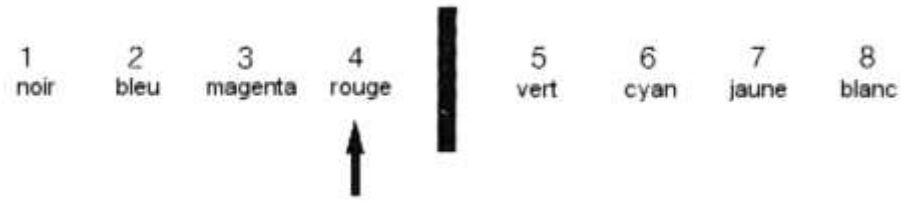


Nous prenons l'élément le plus à gauche, la partie non triée et le comparons à chaque élément de la partie triée, de droite à gauche jusqu'à ce que nous trouvions sa place. Au cours de la comparaison, nous décalons les éléments triés vers la droite, ce qui nous permet d'insérer le nouvel élément.

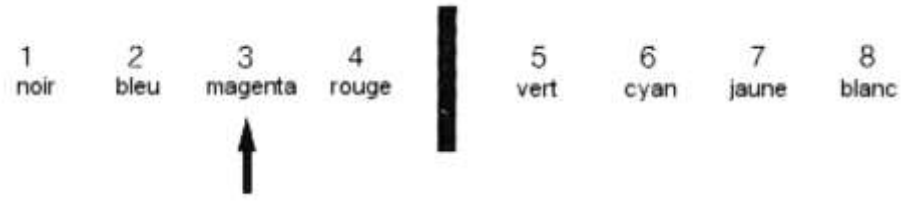
On suppose maintenant que quatre éléments sont triés et nous nous concentrons sur le vert (green), élément le plus à gauche du tableau non trié.



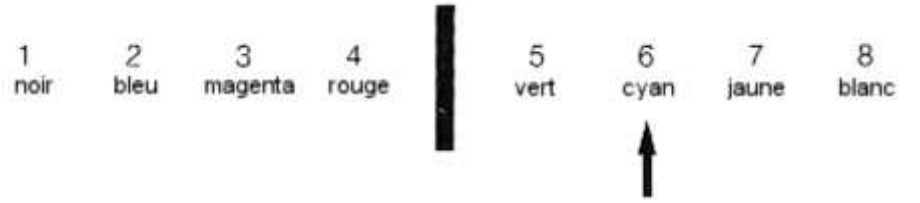
1. Nous mettons le « vert » dans la variable comp$ et mettons une variable p à 5.
2. La variable p indique éventuellement l'endroit où le « vert » doit être placé. Si le « vert » est décalé à gauche alors nous diminuons la valeur de p.
3. Nous comparons le « vert » et le « rouge ». Si le « vert » est supérieur ou égal (plus près de Z) à « rouge », nous sortons et le « vert » reste à sa place. Autrement nous mettons le « rouge » en position 5 et diminuons la valeur de p.



4. Nous continuons avec cette méthode, mais cette fois, nous comparons « vert » et « magenta » et nous obtenons :



5. Finalement nous nous déplaçons vers la gauche et comparons « vert » et « bleu ». Cette fois, il n'est pas nécessaire de changer quelque chose. Nous sortons de la boucle et plaçons green en position 3. Et nous sommes prêts à traiter le sixième élément cyan.



Analyse

1. Premièrement, nous chargeons les couleurs dans un tableau cooleur$(8) et utilisons :
   - comp$ couleur en cours de comparaison
   - p qui indique la position où doit aller la couleur dans comp$
2. Une boucle FOR concentre l'attention sur les positions de 2 à 8 (il y a déjà un élément de trié).
3. Une boucle REPeat fait les comparaisons jusqu'à ce que l'on sache où placer la valeur en cours de comp$.

REPeat compare

IF comp$ n'a pas besoin d'aller plus à gauche: EXIT

établir une couleur à la droite de la position actuelle

décrémenter p

END REPeat compare

4. Après la sortie de la boucle REPeat, la couleur dans comp$ est placée en position p et la boucle FOR se poursuit.

Schéma du programme

1. Déclaration du tableau couleur$
2. Lirecouleur$dans le tableau
3. FOR element 2 TO 8

    LET p = item

    LET comp$ = colour$(p)

        REPeat compare

        IF comp$>=couleurS(p1):EXIT compare

        LET couleurVp) = couleur$(p-l)

        LET p p-1

    END REPeat compare

    LET couleur$(p) = comp$

    END FOR item
4. PRINT couleur$ triées
5. DATA

Les prochains tests décèleraient une erreur. Erreur très fréquente, quand la première donnée n'est pas dans la bonne position dès le départ. Une simple modification des données initiales en

rouge noir bleu magenta vert cyan jaune blanc

montrerait le problème. Nous comparons « noir » avec « rouge » et décrémentons p de valeur 1. Nous essayons ensuite de comparer « noir » et la variable couleur$(p-l) qui vaut couleur$(0) qui n'existe pas.

Ceci est un problème bien connu en informatique, et la solution est de "mettre une sentinelle" à la fin du tableau. Juste avant le début de la boucle REPeat, il faut :

```
LET couleur$ = comp$
```

Par chance, SuperBASIC accepte les indices nuls, sinon il aurait fallu résoudre le problème aux dépens de la lisibilité :

PROGRAMME MODIFIE

```
100 RESTORE
110 REMark tri par insertion
120 DIM couleur$(8,7)
130 FOR item = 1 TO 8:READ couleur$(item)
140 FOR item=2 TO 8
150   LET p=item
160   LET comp$=couleur$(p)
170   LET couleur$(0)=comp$
180   REPeat compare
160     IF comp$>=couleur$(p-1):EXII compare
200     LET couleur$(p)=couleur$(p-1)
210     LET p=p-1
220 END REPeat compare
```

```
230 LET couleur$(p)=comp$
240 END FOR item
250 PRINT "triées..." ! couleur$(1 TO 8)!
260 DATA 'noir', 'bleu', 'magenta','rouge'
270 DATA 'vert','cyan','jaune','blanc'
```

Commentaire

1. Le programme est bon. On peut le tester sur des valeurs plus simples :

   AAAAAAA
   BABABAB
   ABABABA
   BCDEFGH
   GFEDCBA

2. Un tri à insertion n'est pas particulièrement rapide, mais il peut être très utile pour ajouter un petit nombre d'éléments à une liste déjà triée. Il est quelquefois plus pratique de consacrer un peu de temps pour trier quelques éléments, plutôt que beaucoup de temps mais moins souvent à faire un tri complet.

Vous avez maintenant suffisamment de connaissances de base pour suivre le développement d'une manipulation d'un fichier de sept noms et numéros de téléphone.

TRI D'UN FICHIER MICRODRIVE

Pour trier le fichier « tel » en ordre alphabétique sur les noms, il faut le lire, le placer dans un tableau, le trier, et créer un nouveau fichier qui contiendra les noms en ordre alphabétique.

Il ne faut jamais supprimer un fichier avant d'être assuré qu'il est correctement remplacé. Il est plutôt conseillé de copier le premier fichier sous un nom différent, par mesure de sécurité. Voici les traitements nécessaires :

1. Copier le fichier "tel" dans "Tel temp"
2. Mettre le fichier "tel" dans un tableau
3. Trier le tableau
4. Vérifier le tri du tableau
5. Supprimer le fichier "Tel"
6. Créer un nouveau fichier "tel"

Voilà ce que doit faire le programme, et l'on vérifiera le nouveau fichier avec l'instruction :

```
COPY MDV1_Tel to SCR
```

Toutes les vérifications ayant été effectuées, on pourra alors supprimer le fichier temporaire et sauvegarder le fichier trié :

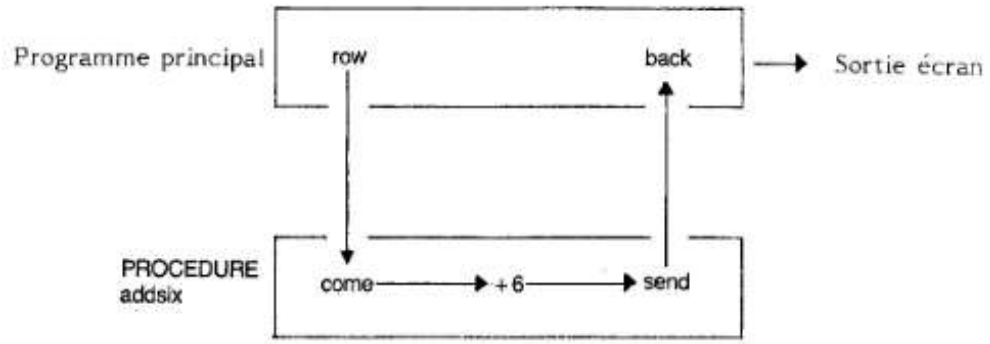
```
DELETE MDV2_tel
COPY MDV1_tel to MDV2_tel
COPY MDV1_tel to SCR
DELETE MDV1_tel_temp
```

TABLEAUX DE PARAMETRES

Dans le programme suivant, nous montrons comment faire passer des tableaux complets du programme principal dans les procédures, et inversement.

A la ligne 40, le tableau 'row' contenant les nombres 1, 2, 3 est transféré dans la procédure 'addsix'. Le paramètre 'corne' reçoit les données et la procédure ajoute six à chaque élément. Le tableau paramètre 'send' contient alors 7, 8, 9.

Ces nombres en provenance du tableau principal, deviennent les valeurs du tableau 'back'. On affichera ces valeurs pour vérifier que les données ont bien été transférées comme on le désirait.




```
100 REMark Pass Arrays
110 DIM row(3),back(3)
120 FOR k = 1 TO 3 : LET row(k) = k
130 addsix row,back
140 FOR k = 1 TO 3 : PRINT ! back(k) !
150 DEFine PROCedure addsix(come,send)
160   FOR k = 1 TO 3 : LET send(k) = come(k) + 6
170 END DEFine
```

Résultat : **7 8 9**

La procédure ci-dessous reçoit un tableau qui contient des données qu'il faut trier. L'élément zéro indique le nombre d'éléments du tableau. Notons qu'il importe peu que le tableau soit numérique ou chaîne. On sait que le système remplacera la chaîne par une donnée numérique si c'est nécessaire.

Autre remarque importante : l'élément du tableau come(0) a deux fonctions :

a) il indique le nombre d'éléments du tableau à trier
b) il contient ensuite l'élément en cours de déplacement.

```
800 DEFine PROCedure sort(come,send)
810   LET num = come(0)
820   FOR item = 2 TO num
830     LET p = item
840     LET come(0) = come(p)
850     REPeat compare
860       IF come(0) >= come(p-1) : EXIT compare
870       LET come(p) = come(p-1)
880       LET p = p - 1
890     END REPeat compare
900     LET come(p) = come(0)
910   END FOR item
920   FOR k = 1 TO 7 : send(k) = come(k)
930 END DEFine
```

Ajoutons les lignes ci-dessous pour tester la procédure sort :

```
10 REMark test de tri
15 RESTORE
20 DIM MOT$(7,3),TRI$(7,3)
25 LET MOT$(0)=7
30 FOR k=1 TO 7:READ MOT$(k)
35 PRINT MOT$!ç
40 sort MOT$,TRI$
50 PRINT TRI$!ç
60 DATA 'VER','FER','MER','PLI',CRI,'TRI','AMI'
```

Sortie: AMI CRI FER MER PLI TRI VER

Commentaire

Ce programme illustre la facilité des manipulations de tableaux en Super-BASIC. Tout ce que vous avez à faire, c'est d'utiliser le nom du tableau pour le passer comme paramètre ou pour l'afficher. Une fois la procédure écrite, vous pouvez l'appeler de n'importe quel programme principal en écrivant :

```
MERGE MDVI sort
```

Vous disposez maintenant d'un nombre de techniques et d'une syntaxe suffisants pour faire un dessin plus complexe à l'écran. On vous propose de représenter les mains de quatre joueurs de cartes. Une main peut se représenter de la manière suivante :

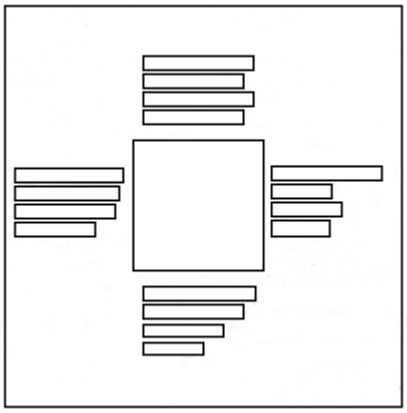
Cœur	A 3 7 D
Carreau	5 9 V
Trèfle	6 10 R
Pique	2 4 D

On améliorera la présentation en dessinant en rouge les carreaux et les cœurs, et en noir les piques et les trèfles. On pourra faire une bande de couleur blanche. On fera le fond en vert et la table de deux couleurs.

Méthode

Puisqu'il faudra envelopper un certain nombre de caractères à imprimer, il est préférable de commencer par positionner ces zones en pixels.

On voit aisément qu'il faudra douze lignes de caractères, un espace entre chaque ligne, et que l'on disposera d'une hauteur d'écran de 256 pixels.



Il est bon de rappeler que les hauteurs possibles pour les caractères sont 10 ou 20 pixels. Il est bien évident que c'est une hauteur de 10 pixels que nous devrons choisir pour avoir un dessin équilibré.

Il faut estimer le nombre de positions de caractères en abscisse. En adoptant la convention que "D" remplace "10", toutes les valeurs de cartes pourront être représentées par un seul caractère. Dans un premier temps, supposons aussi que l'on a un maximum de huit cartes de la même

couleur. Cela requiert un total d'environ dix caractères pour chaque main, et nous aurons en abscisse :

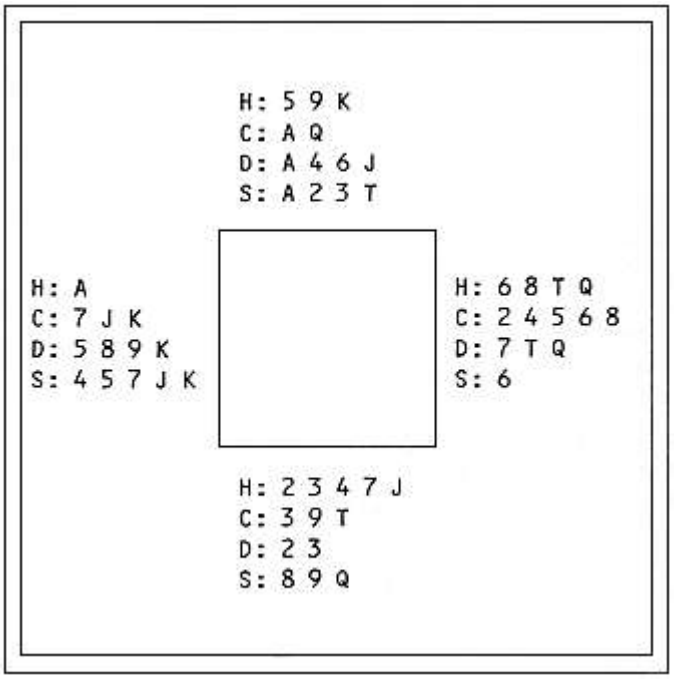
MAIN GAUCHE + LARGEUR DE LA TABLE + MAIN DROITE

avec un espace entre les caractères, cela fait :

20 + LARGEUR DE LA TABLE + 20

Il faut maintenant choisir un mode écran. Le mode 256 pourrait convenir, comme nous le verrons plus tard, mais, dans un premier temps, nous travaillerons en mode 512 pixels. La plus petite largeur possible pour un caractère est six pixels, ce qui nous donne un total de 240 pixels + la largeur de la table. On pourrait équilibrer le schéma en prenant une largeur de table d'environ 240.

On va toutefois faire un essai avec une largeur de table de120 pixels, comme sur le schéma ci-dessous :



FENETRE 440x220 à 35,15
vert , bordure noire de 10 unités

TABLE 100 x 60 à 150,60
damier rouge et vert

MAINS Zones de huit cartes maximum
Position initiale du curseur en
Nord 150,10
Est 260,60
Sud 150,130
Ouest 30,60

Taille des caractères : standard en mode 512

Nombre de pixels entre les lignes : 12

Couleur des caractères : blanc

Couleur des bandes de caractères : rouge pour cœurs et carreaux, noir pour trèfles et piques

Variables

card(52)	changement du nombre de cartes
sort(13)	utilisé pour le tri de chaque main
tok$(4,2)	charge la couleur de carte (cœur, pique, carreau, trèfle)
kmcmh	variables boucles
ran	position aléatoire dans l'échange de cartes
temp	utilisé dans l'échange de cartes
item	insérer une carte dans le tri
dart	pointeur qui indique la position dans le tri
comp	contient un numéro de carte dans le tri
inc	incrément en pixels de rangées de cartes
seat	position de la donne en cours
ac,dn	position du curseur pour les caractères
row	rangée de caractères en cours
lin$	construit une rangée de caractères
max	plus grand numéro de carte
p	point de position de carte
n	numéro de carte en cours

Procédures

Shuffle	distribue 52 cartes
Split	partage les cartes entre quatre mains et appelle sortem pour tirer les cartes de chaque main
Sortem	tire 13 cartes dans un ordre croissant
Layout	procure le fond noir, la bordure et la table
Printem	affiche chaque ligne de symboles de cartes
Getline	fournit une rangéede cartes et convertit leurs numéros en symboles A, 2, 3, 4, 5, 6, 7, 8, 9, 10, V, D, R (V= valet, D = dame , R = roi)

Schéma du Programme

1. Déclarer les tableaux, répartir les types et mettre les 52 nombres dans un tableau card.
2. Distribuer les cartes
3. Les répartir en quatre mains et les trier
4. Ouvrir la fenêtre à l'écran
5. Faire le dessin.
6. Ecrire les quatre mains
7. Fermer la fenêtre de l'écran.

DONNE DE BRIDGE

```
90  RESTORE :MODE 4
100 DIM card(52),sort(13),tok$(4,2)
110 FOR k = 1 TO 4 : READ tok$(k)
120 FOR k = 1 TO 52 : LET card(k) = k
130 shuffle
140 split
150 OPEN #6,scr_440x220a35x15
160 layout
170 printem
180 CLOSE #6
190 DEFine PROCedure shuffle
200   FOR c = 52 TO 3 STEP -1
210     LET ran = RND(1 to c-1)
220     LET temp = card(c)
```

```
230     LET card(c) = card(ran)
240     LET card(ran) = temp
250   END FOR c
260 END DEFine
270 DEFine PROCedure split
280   FOR h = 1 TO 4
290     FOR c = 1 TO 13
300       LET sort(c) = card((h-1)*13+c)
310     END FOR c
320     sortem
330     FOR c = 1 TO 13
340       LET card((h-1)*13+c) = sort(c)
350     END FOR c
360   END FOR h
370 END DEFine
380 DEFine PROCedure sortem
390   FOR item = 2 TO 13
400     LET dart = item
410     LET comp = sort(dart)
420     LET sort(0) = comp
430     REPeat compare
440       IF comp >= sort(dart-1) : EXIT compare
450       LET sort(dart) = sort(dart-1)
460       LET dart = dart - 1
470     END REPeat compare
480     LET sort(dart) = comp
490   END FOR item
500 END DEFine
510 DEFine PROCedure layout
520   PAPER #6,4 : CLS #6
530   BORDER #6,10,0
540   BLOCK #6,100,60,150,60,2,4
550 END DEFine
560 DEFine PROCedure printem
570   LET inc = 12 : INK #6,7
580   LET p = 0
590   FOR seat = 1 TO 4
600     READ ac,dn
610     FOR row = 1 TO 4
620       getline
630       CURSOR #6,ac,dn
640       PRINT #6,lin$
650       LET dn = dn + inc
660     END FOR row
670   END FOR seat
680 END DEFine
690 DEFine PROCedure getline
700   IF row MOD 2 = 0 THEN STRIP #6,0
710   IF row MOD 2 = 1 THEN STRIP #6,2
720   LET lin$ = tok$(row)
730   LET max = row*13
740   REPeat one_suit
750     LET p = p + 1
760     LET n = card(p)
770     IF n >max THEN p = p-1 : EXIT one_suit
780     LET n = n MOD 13
790     IF n = 0 THEN n = 13
800     IF n = 1 : LET ch$ = "A"
```

```
810     IF n >= 2 AND n <= 9 : LET ch$ = n
820     IF n = 10 : LET ch$ = "10"
830     IF n = 11 : LET ch$ = "V"
840     IF n = 12 : LET ch$ = "D"
850     IF n = 13 : LET ch$ = "R"
860     LET lin$ = lin$ & " " & ch$
870     IF p = 52 : EXIT one_suit
880     IF card(p)>card(p+1) : EXIT one_suit
890   END REPeat one_suit
900 END DEFine
910 DATA "C:","T:","K:","P:"
920 DATA 150,10,260,60,150,130,30,60
```

Commentaire

Le programme est exécutable en mode 256, mais les différentes lignes contenant les symboles des cartes peuvent recouvrir la table, ou se trouver en dehors de la bordure de la fenêtre. Une petite modification dans la procédure getline

```
1170 LET lin$ = lin$ & " " & ch$
```

en

```
1170 LET lin$ = lin$ & ch$
```

évitera cette erreur. Les espaces entre les caractères seront supprimés, mais la taille des caractères ayant augmenté, la rangée de caractères sera plus lisible. Ainsi, le programme tourne bien, quel que soit le mode graphique.

Nous avons essayé de montrer comment utiliser SuperBASIC pour résoudre les problèmes. Nous avons montré comment de simples tâches peuvent être résolues d'une manière aisée. Quand la tâche est plus complexe, comme les manipulations de tableaux ou la construction de graphiques à l'écran, SuperBASIC est en mesure d'être employé avec un maximum de clarté.

Si vous êtes débutant, et si vous avez pu suivre l'essentiel de ce guide, vous êtes sur le bon chemin pour programmer correctement. Si vous êtes déjà expérimenté, nous espérons que vous apprécierez et exploiterez les qualités de SuperBASIC.

SuperBASIC peut vous permettre d'exécuter des tâches tellement multiples que nous n'avons pu vous en donner qu'une faible idée dans ce guide. Il ne faut pas oublier les milliers de possibilités qui vous sont offertes, et nous espérons que vous les trouverez profitables, stimulantes et agréables.