SBASIC

MANUEL DE REFERENCE

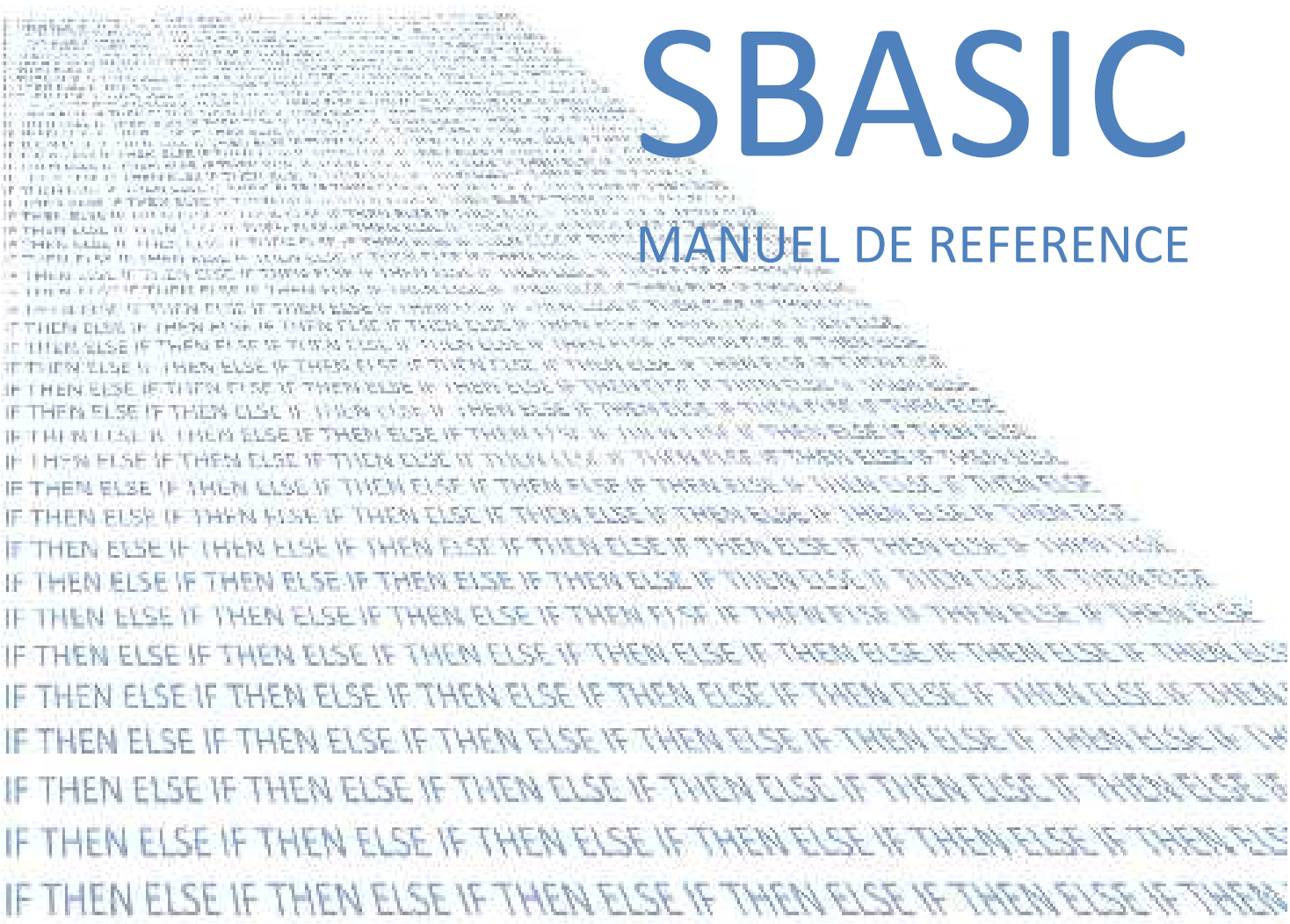

Table des matières

Tableaux

Introduction

SBASIC est un interpréteur très rapide et très complet. Il s'utilise comme la plupart des Basic interprétés :

- **En mode immédiat**, il exécute directement les ordres entrés au clavier, ce qui permet de l'utiliser interactivement comme un calculateur.
- **En mode programme**, les lignes, entrées au clavier, permettent de construire le programme qui sera par la suite lancé au moyen de la commande RUN.

Nous recommandons à l'utilisateur de lire l'ensemble du présent manuel avant de travailler sous SBASIC. Nous y avons inclus de nombreux exemples afin de le rendre plus accessible à la majorité des lecteurs.

SBASIC diffère d'un *Basic* ordinaire par l'adjonction d'un certain nombre de mots-clés et de commandes qui en font un outil puissant de programmation. Citons par exemple, l'usage des procédures, du mode trace, de la renumérotation partielle, de l'édition de la table des variables, etc.

SBASIC apporte ainsi une dimension et un confort nouveaux à la programmation en *Basic*.

Avertissement : Les conventions typographiques utilisées dans ce manuel sont décrites au paragraphe « Syntaxe et abréviations utilisées », page 38.

Première partie : Concepts généraux

SBASIC autorise la manipulation de trois types d'objets différents. Des entiers, des réels et des chaînes de caractères. Les deux premiers types permettent de représenter des nombres alors que le troisième permet de travailler sur du texte.

Tous ces objets sont stockés en mémoire à l'aide d'octets (groupes de 8 bits), chacun d'entre eux pouvant représenter un nombre compris entre 0 et 255 inclus.

1 Types

1.1 Type entier

Le type **Entier** permet de stocker des nombres entiers signés pouvant être codés sur quatre octets, c'est à dire des nombres entiers compris entre -2147483648 et 2147483647. Ils sont représentés en complément à deux, sur 4 octets consécutifs de la mémoire centrale, mais il faut prendre garde au fait que l'ordre dans lequel ces derniers sont utilisés dépend de la machine et même du système utilisé.

1.2 Type réel

Le type **Réel** permet de stocker toutes les autres valeurs numériques. Cette méthode de codage est aussi appelée représentation en virgule flottante. Elle utilise 8 octets par valeur (6,5 pour la mantisse et 1,5 pour l'exposant) et permet de représenter des nombres approximativement compris entre 10^{-300} et 10^{300}.

1.3 Type chaîne de caractères

Le type **Chaîne de caractères** permet de stocker du texte dont l'utilisateur aura besoin pour documenter les résultats d'un programme de calcul ou sur lequel il aura envie de travailler avant de l'éditer. Chaque caractère de texte est alors codé sur un octet (codage ASCII international).

Ce type est souvent appelé type **Chaîne**, ou encore type **Alphanumérique**.

2 Constantes, Variables, Tableaux

2.1 Constantes

Une valeur utilisée sous forme de **Constante** doit être décrite lors de chacune de ses utilisations. Cette description doit respecter les règles suivantes :

2.1.1 Constantes de type entier

Pour une constante de type **Entier** la valeur doit être écrite en décimal et représenter un nombre entier compris entre -2147483648 et 2147483647. Les seuls symboles permis sont donc les chiffres de 0 à 9, l'espace et le signe + ou le signe - (placés en tête).

L'utilisation du point décimal provoque un codage en virgule flottante, même s'il n'est suivi d'aucun chiffre.

Les descriptions suivantes de constantes entières sont correctes :

12 +15 31000 -57 -2545

Les descriptions suivantes sont incorrectes :

12. (codage en virgule flottante à cause du point décimal)
10000000000 (entier supérieur à 2147483647)

2.1.2 Constante de type réel

Pour une constante de type **Réel** la valeur peut, au gré de l'utilisateur, être décrite sous forme décimale ou sous forme scientifique.

Sous forme décimale on utilisera donc les chiffres de 0 à 9, l'espace, le signe + ou le signe - et le point décimal exactement comme dans la vie courante (anglo-saxonne !).

L'écriture sous forme scientifique est recommandée pour des nombres très grands ou très petits en valeur absolue, mais elle peut être utilisée pour toute valeur numérique. En plus des symboles précédents elle tolère la lettre E (majuscule ou minuscule) qui permet de séparer la mantisse de l'exposant. Une constante de type réel doit être comprise entre 10^{-127} et 10^{+127}.

Les descriptions suivantes de constantes réelles sont correctes :

3.14159 -6.25 +12. 6.02E 23 1 E 27

Les deux dernières valeurs sont respectivement égales à : 6.02 x 10^{23} et 10^{27}

Les descriptions suivantes sont incorrectes :

12,5 (utilisation de la virgule)
5.23E10+2 (utilisation du signe + dans l'exposant)

2.2 Constantes de type chaîne de caractères

La description d'une constante de type **Chaîne de caractères** peut utiliser tous les caractères obtenus à partir du clavier, mais elle doit débuter et se terminer par des guillemets (ou des apostrophes) ; ces signes ne font pas partie de la chaîne, ils permettent simplement de la délimiter.

Dans une constante chaîne on peut trouver une ou plusieurs occurrences de guillemets ; une telle constante doit alors être délimitée par des apostrophes et ne peut contenir aucune apostrophe. De même il est possible d'y placer des apostrophes, à condition de la délimiter par des guillemets, mais on ne peut alors y trouver aucun guillemet.

Les descriptions suivantes de constantes chaînes sont correctes :

"Vive le SBASIC" "La lettre 'E' " 'Dites "Bonjour"'

Les descriptions suivantes sont incorrectes :

"A bas le SBASIC 'lettre" 'la lettre "A" et l'apostrophe'

2.3 Variables

On utilise une variable pour manipuler des valeurs qu'on ne veut pas ou qu'on ne peut pas décrire lors de chacune de ses utilisations. Ce peut être parce que la valeur est longue et donc fastidieuse à décrire, ou parce qu'elle dépend du cheminement suivi lors de l'exécution du programme et que le programmeur n'est pas capable de la décrire lors de l'écriture de ce dernier.

Dans ce cas la valeur en cause est manipulée par l'intermédiaire d'un identificateur appelé *« nom de variable »*. Le programmeur se trouve dans la situation d'un manutentionnaire qui travaille sur des boîtes dont il ne connaît pas vraiment le contenu, mais qu'il repère par un nom ou un numéro écrit sur cette dernière (nom de variable) et qui seul lui permet de distinguer une boîte d'une autre. Ce qui se trouve dans la boîte y a été mis par quelqu'un d'autre à un autre moment (*affectation de la variable*).

Pour pouvoir utiliser des variables, il faut connaître les règles qui régissent la formation de leurs noms et la façon de leur affecter une valeur.

2.3.1 Noms de variables

Un **nom de variable** se construit en respectant les règles suivantes :

- Un nom de variable doit être une suite d'au plus 255 caractères pris parmi les lettres de l'alphabet, le caractère souligné « _ », les chiffres et les symboles réservés « % » et « $ ».
- Le premier caractère doit être une lettre de l'alphabet, les suivants peuvent être des lettres, « _ » ou des chiffres, seul le dernier peut être « % » ou « $ ».
- Deux suites différentes permettent de gérer des variables différentes. Tous les caractères sont discriminatoires. Si des lettres minuscules sont utilisées lors de la saisie, elles sont automatiquement converties en majuscules. La chasse des lettres n'est donc pas discriminatoire.
- Un nom de variable ne doit pas commencer par un mot réservé du SBASIC. Voir la liste des mots réservés se trouvant en « Annexe 2 : Liste des codes d'erreurs réservés ».
- Les variables dont le nom se termine par le symbole « % » ne pourront être utilisées que pour contenir des valeurs de type entier.
- Celles dont le nom se termine par « $ » devront avoir un contenu de type chaîne de caractères.
- Les autres variables permettent de manipuler les valeurs réelles.

Exemples de noms de variables syntaxiquement corrects :

- Variables de type entier :
  KILO% N% NOMBRE_PAR_KILO%
- Variables de type réel :
  VITESSE X POIDS_AU_LITRE
- Variables de type chaîne :
  TXT$ A$ LIGNE_COURANTE$

Exemples de noms de variables incorrects :

1AB, $, événement	(Ne commence pas par une lettre majuscule)
A%B$, A/B	(Comporte des symboles interdits)
TABLEAU, TOTAL%	(Commence par un mot réservé, « TAB » ou « TO »)

2.3.2 Affectation

La première chose à faire lorsque l'on utilise une variable est de lui *Affecter* une valeur, c'est à dire, en poursuivant la comparaison entre variables et boîtes, de mettre quelque chose dans la boîte sur laquelle est inscrit le nom de la variable.

Une telle opération se fait à l'aide d'une instruction d'affectation qui peut être effectuée par l'un des mots réservés LET, INPUT ou READ entièrement décrits dans la suite de ce manuel. Toutefois, on peut dès à présent brièvement décrire l'utilisation de LET et de INPUT.

LET *var* = *exp*

permet d'affecter à la variable de nom *var* la valeur de l'expression *exp* qui doit être de même type.

Par exemple :

LET A=1 affecte à la variable A la valeur 1

```
LET NOM$="toto"        affecte à la variable A$ la valeur "toto"
LET VI=12.5            affecte à la variable réelle VI la valeur 12,5
```

Nous rencontrons ici le premier exemple de la syntaxe que nous utiliserons tout au long de cet ouvrage pour décrire les différentes instructions de SBASIC. L'utilisation de l'italique permet de représenter un objet SBASIC que l'utilisateur devra remplacer lors de l'écriture d'un programme par un objet correspondant à cette description.

Ici, nous avons utilisé *var* qui doit être remplacé par un nom de variable SBASIC syntaxiquement correct. De même *exp* devra, lui, être remplacé par une expression quelconque dont l'exemple le plus simple est une constante.

Il est possible, dans les instructions précédentes, d'omettre le mot réservé LET et d'écrire respectivement pour chacune des trois affectations précédentes :

```
A=1
NOM$="toto"
VI=12.5
```

Nous verrons ultérieurement que l'on peut, à droite du signe « = » utiliser des expressions plus complexes que de simples constantes, mais à gauche de ce signe **on doit uniquement trouver un nom de variable**.

Par exemple, l'instruction :

```
LET A+1 = 10
```

est tout à fait incorrecte et ne permet pas d'affecter à la variable A la valeur 9 comme inciterait à le penser toute allusion à l'équation algébrique *« x +1= 10 »*.

Autre exemple :

num-ligne INPUT *var*

Cette instruction arrête l'exécution du programme, affiche dans la fenêtre SBASIC un point d'interrogation et attend que l'utilisateur frappe une entrée au clavier. La ligne entrée par l'utilisateur est alors interprétée par SBASIC et la valeur correspondante affectée à la variable *var*.

Si la variable *var* est de type numérique (entier ou réel) et que l'utilisateur frappe des caractères qui ne peuvent intervenir dans la définition d'une constante de ce type, le programme signale une erreur.

Par exemple le programme :

```
10 PRINT "Quel est votre nom ? ";
20 INPUT NOM$
30 PRINT "Bonjour ";NOM$
```

permet à l'utilisateur de cette machine de se faire saluer très poliment par son ordinateur préféré ; il lui suffit d'entrer ces quelques lignes, puis de les exécuter en frappant au clavier la commande RUN.

Signalons que l'entrée au clavier doit se terminer par la frappe d'un caractère Retour-Chariot correspondant à la touche : ↵

2.3.3 Tableaux

Lorsqu'un programme nécessite l'utilisation d'une liste de variables de même type et que chacune d'entre elles peut être repérée par son rang dans la liste, il est pratique et souvent indispensable de se servir d'un **Tableau**, encore appelé *variable indicée*.

2.3.4 Notion de tableau à une dimension

Par exemple un enseignant qui désire travailler sur les notes obtenues par ses élèves lors d'une composition utilisera une liste, un tableau, dont chacun des éléments contiendra la note d'un élève. La liste de classe étant supposée, ce qui est fréquent, écrite dans l'ordre alphabétique, chaque élève sera simplement repéré par son rang dans cette liste.

En supposant avoir une classe de 5 élèves ayant les noms suivants :

ANDRE
JACQUES
JEAN
PAUL
PIERRE

La note de l'élève n° 1 (ANDRE) correspondra à l'élément 1 du tableau,
La note de l'élève n° 2 (JACQUES) correspondra à l'élément 2 du tableau
La note de l'élève n° 3 (JEAN) correspondra à l'élément 2 du tableau
et ainsi de suite

2.3.5 Dimensionnement d'un tableau à une dimension

Avant toute utilisation d'un tableau à une dimension permettant de gérer une liste de variables de même type, il faut déclarer ce tableau en indiquant son nom et en le dimensionnant. Cette opération permet à SBASIC de se préparer à la gestion d'une telle liste.

Pour dimensionner un tableau on utilise l'instruction DIM, décrite complètement dans la suite de ce manuel, mais dont on peut donner ici brièvement la syntaxe.

DIM *var*(*exp-num*)

var est le nom du tableau. La construction de ce nom obéit aux mêmes règles que celles régissant la construction d'un nom de variable.

Si chaque élément du tableau est destiné à contenir une valeur de type entier, le nom *var* doit de terminer par le symbole réservé « % ».

Si chaque élément du tableau est destiné à contenir du texte, le nom *var* doit se terminer par le symbole réservé « $ ».

exp-num est un entier indiquant le nombre maximum d'éléments que l'on désire ranger dans le tableau.

Aucune opération globale d'affectation ou d'édition ne peut être faite en utilisant le nom du tableau. Mais chacun de ses éléments peut être manipulé exactement comme une variable de type correspondant.

2.3.6 Utilisation d'un tableau a une dimension

En reprenant l'exemple de la classe de 5 élèves, il est possible d'utiliser deux tableaux parallèles, l'un permettant de ranger les noms des élèves, l'autre contenant leurs notes.

Ces tableaux sont dimensionnés de la façon suivante :

```
10 N%=5
20 DIM NOM$(N%)
30 DIM NOTE(N%)
```

Pour entrer les noms et les notes des différents élèves de la classe il suffit d'exécuter les lignes suivantes :

```
100 FOR I%=1 TO N%
110 REM
120 REM Affectation d'un contenu à la variable NOM$(I%)
130 PRINT "Nom de l'élève ";I% ;
140 INPUT NOM$(I%)
150 REM
160 REM Affectation d'une note à l'élève de rang I%
170 PRINT "Note de l'élève ";NOM$(I%);
180 INPUT NOTE(I%)
```

```
190 NEXT I%
```

Pour obtenir maintenant une édition de ces notes dans l'ordre alphabétique il suffit d'exécuter les instructions suivantes :

```
200 FOR I%=1 TO N%
210 PRINT NOM$(I%); " a pour note "; NOTE(I%)
220 NEXT I%
```

Pour ce petit programme on aurait évidemment pu utiliser 5 variables de type chaîne de caractères destinées à contenir les noms N1$, N2$, N3$, N4$, N5$ ainsi que 5 variables de type réel destinées à contenir les notes, N1, N2, N3, N4, N5.

Toutefois la méthode utilisée présente deux avantages. Tout d'abord elle évite d'écrire 5 fois chacune des lignes mettant en jeu les éléments NOM$(I%) et NOTE(I%), à savoir les lignes 110, 120, 130, 210. Ensuite elle donne un programme aisément modifiable et adaptable à une classe plus conforme à la réalité. Il suffit par exemple de modifier l'affectation de la ligne 10, en la remplaçant par :

```
10 N%=25
```

on obtient alors un programme valable pour une classe de 25 élèves.

2.3.7 Notion de tableau à deux dimensions

Si le même enseignant désire travailler sur l'ensemble des notes de ses élèves pour un mois donné, il devra disposer, pour chacune des quatre interrogations du mois, de la liste des notes de ses élèves.

Il pourrait évidemment utiliser un tableau pour chacune des listes de notes :

NOTE1(*) contenant les notes de tous les élèves pour la première interrogation,
NOTE2(*) contenant les notes de tous les élèves pour la deuxième interrogation,
Et ainsi de suite.

Mais on retombe alors dans le double travers signalé à la fin du paragraphe précédent concernant la concision d'écriture du programme et les possibilités d'adaptation à d'autres cas de figure semblables.

Il est préférable d'utiliser un tableau à deux dimensions, qui permettra, comme la page d'un cahier de notes, de stocker l'ensemble des notes du mois. L'un des indices permettra de repérer le nom de l'élève concerné (la ligne du cahier) et l'autre d'indiquer de quelle interrogation il s'agit (la colonne du cahier).

2.3.8 Dimensionnement et utilisation d'un tableau double

Un tableau double se déclare par une instruction de dimensionnement semblable à celle utilisée pour un tableau à une dimension :

DIM *var*(*exp-num-1*,*exp-num-2*)

var est le nom du tableau qui doit être construit comme celui d'un tableau à une dimension en fonction du type de valeurs que l'on désire y stocker.

exp-num-1 est le nombre maximum de lignes du tableau.

exp-num-2 est le nombre maximum de colonnes du tableau.

L'utilisation du tableau nécessaire à la gestion de l'ensemble des notes de l'exemple précédemment décrit imposera la déclaration d'un tableau de notes doublement indicé grâce à l'instruction :

```
50 DIM NOTE_MOIS(5,4)
```

Dans ce tableau :

NOTE_MOIS(2,3) est la note obtenue par JACQUES à l'interrogation 3,
NOTE_MOIS(4,1) est la note obtenue par PAUL à l'interrogation 1.

Il faudra utiliser une double boucle pour entrer tous les éléments du tableau, mais l'utilisateur aura ensuite la possibilité de d'établir, pour chaque élève, la moyenne des notes du mois et de calculer pour

chacune des interrogations la moyenne des notes de la classe.

De même que pour les tableaux à une dimension, aucune opération (affectation, édition, …) ne peut être faite globalement en invoquant seulement le nom du tableau, mais chacun de ses éléments peut être utilisé dans les mêmes conditions qu'une variable de type correspondant.

SBASIC supporte un nombre quelconque de dimensions à condition que le nombre total d'éléments ne dépasse pas 536870911.

3 Operateurs et expressions

Ayant défini précédemment les objets élémentaires (constantes et variables) sur lesquels travaillait SBASIC, nous allons maintenant définir les opérateurs qui permettent de construire d'autres objets plus complexes, les expressions dont la valeur peut être un nombre ou une chaîne de caractères, ainsi que des relations entre ces objets.

Par exemple :

A désignant une variable numérique et A$ une variable chaîne, on pourra être amené dans le cours d'un programme à utiliser les expressions :

```
12 + EXP(A)
PI + LEN(A$)
```

que l'on pourra désigner respectivement par *exp-num-1* et *exp-num-2*. En appliquant à ces mêmes expressions les règles de remplacement que nous avons vues au sujet des variables, nous voyons que la première, par exemple, est obtenue à partir de l'expression formelle :

const-num + EXP(*var-num*)

en y remplaçant respectivement :

const-num par 12
var-num par A

De même, les expressions prenant des valeurs de type chaîne de caractères seront désignées par *exp-chaîne* éventuellement suivie d'un chiffre, ce qui pourra donner *exp-chaîne-1* ou *exp-chaîne-2* lorsque l'on désirera utiliser deux éléments de ce type.

On parlera, par exemple, de l'expression formelle :

var-chaîne = MID$(*const-chaîne*,*var-num-1*,*const-num*)

qui pourra devenir pour une utilisation particulière :

```
EXTRAIT$ = MID$("SIROS SBASIC",I%,3)
```

3.1 Opérateurs

Trois catégories d'opérateurs sont disponibles.

Les premiers opèrent sur des objets (expressions numériques ou alphanumériques) et permettent de construire d'autres objets plus complexes, c'est-à-dire d'autres expressions plus complexes. Ce sont les **Opérateurs arithmétiques** et **l'Opérateur de concaténation de chaînes**.

Les seconds opèrent sur des objets et permettent de former des relations entre ces objets. Ce sont les **Opérateurs de relations**.

Enfin, la dernière catégorie est celle des **Opérateurs logiques** qui peuvent opérer sur des relations pour donner d'autres relations, et qui peuvent opérer sur des valeurs numériques pour donner d'autres valeurs numériques.

3.1.1 Operateurs arithmétiques

La première catégorie d'opérateurs, avec laquelle nous sommes le plus familiarisés, est celle des opérateurs arithmétiques. Ces opérateurs permettent de relier deux expressions numériques, pour

former des expressions numériques de plus en plus complexes.

Ce sont :

Nom	Symbole	Exemple	Explication
Addition	+	X+Y	Ajoute X à Y
Soustraction	-	X-Y	Soustrait Y de X
Multiplication	*	X*Y	Multiplie X par Y
Division	/	X/Y	Divise X par Y[1]
Exponentiation	**	X**Y	Elève X à la puissance Y[2]
Modulo	MOD	X MOD Y	Calcule X modulo Y

Tableau 1 – Opérateurs arithmétiques

Dans chacune des phrases précédentes il faut évidemment lire *contenu de X* et *contenu de Y* au lieu de X et de Y. Ce n'est que pour des questions de présentation que nous nous sommes permis ce raccourci.

De plus on pourra évidemment remplacer chacune des variables X et Y par une variable entière ou même par une expression numérique quelconque *exp-num-1* ou *exp-num-2*.

Lorsque deux valeurs de type différent sont concernées, un entier et un flottant (nombre réel) par exemple, l'entier est d'abord converti en flottant et l'opération est effectuée. Seul l'opérateur d'exponentiation fait exception à cette règle.

3.1.2 Operateur de concaténation de chaînes

De même que les opérateurs précédents permettent de construire des expressions dont la valeur est de type numérique, il existe un opérateur permettant de construire des expressions alphanumériques plus complexes.

Cet opérateur, représenté par le signe « + », permet de relier deux expressions de type chaîne de caractères pour en obtenir une troisième du même type. L'expression suivante :

exp-chaîne-1 + exp-chaîne-2

aura pour valeur la chaîne obtenue en concaténant (mettant bout à bout) les deux chaînes dans l'ordre donné.

Attention :
contrairement à l'opération que ce symbole « + », représente lorsqu'il agit sur des expressions numériques, la concaténation de chaînes n'est pas commutative. Les valeurs de *exp-chaîne-1 + exp-chaîne-2* et de *exp-chaîne-2 + exp-chaîne-1* ne sont en général pas égales.

Par exemple si A$ et B$ contiennent respectivement "Sbasic " et "SIROS ", la valeur de A$+B$ est "Sbasic SIROS ", alors que celle de B$+A$ est "SIROS Sbasic ".

---

[1] Il faut évidemment que le contenu de Y (ou de <exp-num-2>) soit non nul !
Si <exp-num-1> et <exp-num-2> sont de type entier, le résultat donné est le quotient entier de <exp-num-1> par <exp-num-2>.
Il ne faudra pas s'étonner de trouver 1 comme valeur de l'expression 3/2. En revanche 3./2 ou 3/2. donnera le résultat escompté : 1.5, le point décimal de chacune de ces expressions forçant le type réel du résultat.

[2] Plus généralement on peut former l'expression <exp-num-1>**<exp-num-2>. Si le contenu de <exp-num-2> est un entier, le contenu de <exp-num-1> peut être quelconque, c'est alors l'algorithme de multiplication qui est utilisé. Mais dans les autres cas, le contenu de <expr-num-l> doit être positif car la valeur correspondante sera calculée par la formule :
EXP (<exp-num-2> * LOG (<exp-num-1>))
Il faut évidemment que le contenu de Y (ou de <exp-num-2>) soit non nul !

3.1.3 Operateurs de relation

Les opérateurs de relation permettent de construire des expressions logiques élémentaires qui pourront être utilisées par exemple dans un test introduit par le mot réservé IF.

Les expressions logiques élémentaires se construisent en comparant, à l'aide de ces opérateurs, des expressions de même type, numérique ou alphanumérique. Les six symboles de relation reconnus par SBASIC sont :

Symbole	Exemple	Signification
=	*exp-1* = *exp-2*	*exp-1* égal à *exp-2*
<>	*exp-1* <> *exp-2*	*exp-1* différent de *exp-2*
<	*exp-1* < *exp-2*	*exp-1* inférieur à *exp-2*
>	*exp-1* > *exp-2*	*exp-1* supérieur à *exp-2*
<=	*exp-1* <= *exp-2*	*exp-1* supérieur ou égal à *exp-2*
>=	*exp-1* >= *exp-2*	*exp-1* inférieur ou égal à *exp-2*

Tableau 2 – Opérateurs de relation

Lorsqu'il s'agit d'expressions numériques, l'interprétation ne pose aucun problème. Pour les expressions de type alphanumérique, la relation d'ordre utilisée est l'ordre alphabétique des chaînes de caractères. La valeur de l'expression alphanumérique *exp-chaîne-1* est supérieure à celle de l'expression alphanumérique *exp-chaîne-2* si la chaîne correspondant à la première s'écrit après la chaîne correspondant à la seconde dans le grand dictionnaire des chaînes alphanumériques.

Par exemple, si A$ contient « SIROS », la relation A$ <= "SBASIC" est une relation fausse, car dans l'ordre alphabétique « SIROS » vient après « SBASIC ».

Ces relations logiques élémentaires sont souvent combinées entre elles grâce aux opérateurs logiques (Voir le paragraphe 3.1.4 « Operateurs logiques ») pour donner des expressions logiques plus complexes qui seront désignées génériquement dans la suite par *exp-logique*.

3.1.4 Operateurs logiques

La dernière classe d'opérateurs est celle des opérateurs logiques (AND, OR, NOT). Ils permettent de relier des valeurs de type entier ou des expressions logiques.

3.1.4.1 Operateurs logiques entre valeurs de type entier

Lorsqu'ils relient des valeurs de type entier, les opérateurs logiques permettent de réaliser des opérations bit à bit, afin de former des expressions de type numérique.

Si, par exemple, A% et B% contiennentrespectivement les quantités binaires suivantes :

A% contient 11001011111101101100101111110110 (soit -873018378 en décimal)
B% contient 01110101111001000111010111100100 (soit 1977906660 en décimal)

Alors :

A% AND B% a pour valeur 01000001111001000100000111100100 (1105478116)
A% OR B% a pour valeur 11111111111101101111111111110110 (-589834)
NOT A% a pour valeur 00110100000010010011010000001001 (873018377)

De façon plus générale, les effets des opérateurs logiques sont décrits ci-dessous :

Opérateur	Exemple	Action
NOT	NOT X%	Cet opérateur transforme la valeur de chaque bit en son complément (les 1 sont remplacés par des 0 et les 0 par des 1).

AND	X% AND Y%	Le résultat de cette opération est d'affecter, à chaque bit du résultat, un « 1 » si chacun des bits de même rang vaut « 1 » dans X% et Y%.
OR	X% OR Y%	L'opération met à « 1 » chaque bit du résultat si l'un ou l'autre des bits de même rang dans X% ou Y% vaut « 1 ».

Tableau 3 – Opérateurs de relation

On peut évidemment, dans chacun des trois exemples précédents, remplacer l'une ou l'autres des variables X% et Y% par une expression numérique.

Dans la mesure où ces opérateurs sont définis par leur effet sur des quantités entières, la représentation interne en virgule flottante de certaines variables et constantes doit d'abord être convertie en entiers. Cette conversion est réalisée automatiquement par l'interpréteur SBASIC.

Ces opérateurs peuvent s'utiliser conjointement aux opérateurs arithmétiques dans des expressions numériques.

Si l'on veut par exemple, pour le contenu d'une variable A% faire la somme de son quotient par 16 et de son reste modulo 16, puis affecter la valeur correspondante à la variable B%, on pourra écrire :

```
B% = (A%/16) + (A% AND 15)
```

Remarque : Pour l'extraction d'un modulo, on ne peut utiliser la formule *var-num* AND *exp-num* que si la valeur de *exp-num* est de 2^n -1, n étant un entier compris entre 1 et 31.

3.1.4.2 Operateurs logiques entre expressions

Les opérateurs logiques AND, OR, NOT peuvent aussi, et c'est en fait leur utilisation la plus fréquente, opérer sur des expressions logiques. Ils permettent alors de former des expressions logiques plus complexes, qui seront le plus souvent testées grâce à l'instruction IF.

Si *exp-logique-1* et *exp-logique-2* représentent des expressions logiques, on peut former :

Opération	**Résultat**
NOT *exp-logique-1*	Vrai si et seulement si *exp-logique-1* est fausse.
exp-logique-1 AND *exp-logique-2*	Vrai si et seulement si *exp-logique-1* et *exp-logique-2* sont simultanément vraies.
exp-logique-1 OR *exp-logique-2*	Vrai dès que l'une au moins des deux expressions *exp-logique-1* ou *exp-logique-2* est vraie.

Tableau 4 – Opérateurs logiques entre expressions

Dans l'exemple :

```
40 IF (A$="FIN" AND I%=10) THEN GOTO 100
```

le programme se branchera à la ligne 100 si la variable A$ contient la valeur « FIN » et si I% contient la valeur 10. Les parenthèses ne sont pas obligatoires, elles ont simplement été placées pour mieux mettre en évidence l'expression logique testée.

Toutefois les deux utilisations des opérateurs logiques peuvent être combinées, car à chaque expression logique on peut, à un instant donné, associer une valeur numérique entière qui vaut 0 si sa valeur logique est FAUX et -1 si elle est VRAI.

On pourra donc par exemple écrire :

var-num = exp-logique-1 AND *exp-logique-2*

ce qui permettra d'affecter à *var-num* la valeurs -1 si *exp-logique-1* et *exp-logique-2* sont toutes deux vraies. L'expression précédente permet évidemment de remplacer avantageusement :

IF *exp-logique-1=exp-logique-2* THEN *var-num*=-1 ELSE *var-num*=0,

mais *exp-logique-1* AND *exp-logique-2* peut aussi s'utiliser de la même façon que n'importe quelle autre expression de type numérique.

3.2 Priorité entre les operateurs

Quand SBASIC évalue une expression contenant un mélange d'opérateurs mathématiques, il effectue d'abord l'exponentiation, prend en compte ensuite tous les signes moins *unaires* (comme -3.4 ou -A). Il effectue ensuite les multiplications et divisions, et enfin les additions et soustractions. Lorsque des opérateurs de priorité égale sont rencontrés, c'est celui situé le plus à gauche qui est d'abord exécuté, car SBASIC évalue les expressions de gauche à droite.

L'ordre peut toutefois être modifié par l'utilisation de parenthèses. SBASIC évalue d'abord les quantités entre les parenthèses les plus internes. Les parenthèses doivent être utilisées à chaque fois qu'il y a un doute au sujet de l'évaluation de l'expression.

Lorsque deux valeurs de même type sont concernées, l'opérateur approprié est appelé. Par exemple, lorsque l'on multiplie deux entiers, la routine de multiplication d'entiers est appelée. Si deux nombres en *virgule flottante* doivent être soustraits, la routine de soustraction en *virgule flottante* est appelée.

Les priorités entre opérateurs sont indiquées dans le tableau ci-dessous. Les opérateurs sont listés par ordre de priorité décroissante. Les opérateurs de même priorité sont évalués de gauche à droite.

Opérateur	Priorité
( )	Expressions placées entre parenthèses
**	Exponentiation
-	Signe moins (unaire)
* /	Multiplication et division
MOD	Modulo
+-	Addition et soustraction
<> = >= <=	Opérateurs de relation
NOT	Opérateur NOT
AND	Opérateur AND
OR	Opérateur OR

Tableau 5 - Priorités entre operateurs

Il faut toutefois se rappeler que si, en plus de ces opérateurs, des fonctions interviennent dans l'expression à évaluer, ces dernières ont une priorité encore plus élevée que les parenthèses.

4 Utilisation de SBASIC en mode direct et en mode programme

4.1 Instructions

L'interpréteur SBASIC permet d'analyser et d'exécuter des instructions qui sont construites à partir des mots réservés dont le mode d'emploi détaillé est donné dans la deuxième partie de ce manuel.

A titre d'exemple on peut donner des instructions d'affectation :

```
LET A = 12 * PI
B = (A + 5) * 12.5
TITRES = "Documentation SBASIC"
```

ou des instructions d'impression :

```
PRINT "Dernier chapitre"
```

```
PRINT "Page numéro "; NP%
```

4.2 Mode direct

Ces instructions peuvent s'utiliser en MODE DIRECT, c'est à dire telles que nous les avons écrites ci-dessus ; elles sont alors directement analysées et exécutées par SBASIC. On dit alors qu'il s'agit de **Commandes**.

Si l'on tape au clavier l'avant dernier exemple donné plus haut, suivi d'un Retour-Chariot indiquant à SBASIC que l'on a terminé d'entrer une commande, ce dernier l'exécutera immédiatement : il affichera la chaîne de caractères « Dernier chapitre » dans la fenêtre SBASIC.

Il est possible d'entrer en une seule fois plusieurs commandes que SBASIC exécutera les unes après les autres, il suffit de les séparer par le signe deux points « : ». Il faut évidemment finir par un Retour-Chariot indiquant à SBASIC que l'utilisateur a terminé sa ligne de commandes. Par exemple on peut taper :

```
NP%=10 : PRINT "Page "; NP%
```

ce qui affecte la valeur 10 à la variable entière NP%, puis affiche dans la fenêtre SBASIC le libellé « Page 10 ».

4.3 Mode programme

On peut également entrer des instructions que SBASIC n'exécute pas immédiatement, mais qu'il mémorise sous forme de **lignes de programme** qui seront exécutées ultérieurement. C'est ce que l'on appelle le mode programme.

Avant tout travail dans ce mode, il est conseillé d'exécuter la commande NEW (ne pas oublier le Retour-Chariot final) qui efface toute ligne de programme qui aurait pu être entrée accidentellement par l'utilisateur courant de la machine, ou sciemment par un utilisateur antérieur.

Il est par exemple possible d'entrer le programme suivant :

```
10 PRINT "Quel est votre nom "
20 INPUT NOM$
30 PRINT "Bonjour "; NOM$
```

Pour ce faire il suffit à l'utilisateur de frapper au clavier les lignes telles qu'elles sont écrites, en les terminant toujours par un Retour-Chariot. S'il commet une erreur lors de l'entrée de l'une d'entre elles, il lui suffit de la frapper à nouveau.

Contrairement à ce qui s'est passé dans l'exemple précédent, aucun message ne s'affiche lorsque l'utilisateur frappe le Retour-Chariot terminant la première et la dernière ligne. C'est que l'on vient de créer 3 lignes de programme. Pour **Exécuter** ce programme, l'utilisateur doit frapper au clavier la commande RUN (suivi d'un Retour-Chariot) qui débutera l'exécution du programme.

Ce dernier commencera par demander le nom de l'utilisateur puis le saluera poliment. Si l'utilisateur ne se lasse pas de cette politesse et qu'il désire un nouveau salut, il lui sera inutile de réécrire les lignes de programmes, il lui suffira de frapper à nouveau la commande RUN, c'est ce qui fait tout l'intérêt du mode programme.

4.4 Sauvegarde et chargement d'un programme

Malheureusement ce programme phénoménal se trouve dans la mémoire centrale de la machine qui est constituée de mémoire vive (RAM) dans laquelle on peut écrire facilement, mais qui s'efface tout aussi facilement à la moindre coupure de courant. Si l'utilisateur veut conserver son programme, il lui suffit de l'enregistrer sur disque en utilisant la commande :

```
SAVE "PREMIER"
```

qui lui permettra de le sauvegarder sur disque dans un fichier nommé « PREMIER ». En fait le nom complet du fichier, tel qu'il pourra apparaître dans un répertoire de fichiers, sera « PREMIER.BAS », mais l'extension « .BAS » est gérée

automatiquement par le système et l'utilisateur n'a pas à s'en préoccuper pour l'instant.

Après une interruption du courant ou même l'utilisation de la commande NEW, l'utilisateur pourra recharger les trois lignes du programme précédent en tapant la commande :

```
LOAD "PREMIER"
```

S'il utilise alors la commande :

```
LIST
```

qui permet de LISTer le programme se trouvant en mémoire centrale, il pourra vérifier qu'il a effectivement retrouvé le programme initial.

4.5 Utilisation conjointe du mode programme et du mode direct

Il est possible de panacher les deux modes d'utilisation de l'interpréteur SBASIC. Même après l'exécution du programme précédent la variable NOM$ n'a pas « perdu » son contenu et l'utilisateur peut taper l'ordre :

```
PRINT NOM$
```

qui permettra de vérifier son contenu. Cette dernière possibilité, dans laquelle réside une grande partie de l'intérêt des langages interprétés, offre de grandes facilités lors de la mise au point et le dépannage des programmes écrits en SBASIC. Elle est aussi utilisable lorsque le programme a été interrompu par une instruction STOP, l'appui simultané sur les touches Ctrl et C (notée Ctrl-C dans la suite de ce manuel) ou la survenue d'une erreur.

4.6 Description d'une ligne de programme SBASIC

Chaque ligne de programme BASIC commence par un numéro de ligne. La ligne peut contenir une ou plusieurs instructions séparées par le caractère « : » et s'achève par un Retour-Chariot.

La longueur d'une ligne ne peut dépasser 255 caractères. Les lignes peuvent être numérotées de 1 à 2147483647 et chacune d'entre elles a un numéro unique. Lorsque SBASIC exécute un programme, il interprète les lignes dans l'ordre croissant de leurs numéros.

Lorsque l'on écrit un programme, il est conseillé de numéroter les lignes de 10 en 10, ou de 20 en 20 afin de pouvoir, éventuellement par la suite, en insérer facilement de nouvelles.

On peut aussi laisser des espaces pour rendre plus aisée la lecture d'un programme. Dans les exemples suivants, les instructions 30 et 40 sont identiques, mais la ligne 40 est plus facile à lire. Il est recommandé de mettre un espace entre un mot clé et un nom de variable. Un espace est obligatoire entre un nom de variable de type *virgule flottante* et un mot clé.

Exemples d'utilisation d'espaces :

```
30X=32*X/3+7.3*X/2+6.54*X-.1
40 X=  32*X/3  +  7.3*X/2  +  6.54*X  -  .1
```

Exemple de ligne où un espace est obligatoire :

```
100 IF IND=0 THEN PRINT VITESSE ELSE PRINT TEMPS
```

le seul espace obligatoire est celui placé entre VITESSE et ELSE ; les autres sont facultatifs.

Exemple comportant plusieurs instructions par ligne :

```
120 INPUT "VITESSE, TEMPS"; V,T : D = V*T : PRINT "DISTANCE ="; D
```

4.7 Restrictions sur l'utilisation en modes direct et programme

La majeure partie des instructions du SBASIC peut s'utiliser en mode direct et en mode programme. Toutefois, certaines d'entre elles ne peuvent s'utiliser que dans l'un des deux modes.

4.7.1 Les commandes pures

Voici la liste des commandes ne pouvant s'utiliser qu'en mode direct. Elles ne peuvent donc figurer

dans les lignes d'un programme.

Commande	Action
BREAK ON	Valide la possibilité d'arrêter un programme en cours de traitement au moyen de la combinaison de touches Ctrl-C. Cette option est prise par défaut au chargement du SBASIC.
BREAK OFF	Invalide l'action Ctrl-C.
CONT ou CONTINUE	La commande CONTinue est utilisée pour faire redémarrer un programme arrêté par une instruction STOP ou un appui sur Ctrl-C.
EDIT	Pour modifier une ligne de programme, il suffit de taper EDIT suivi du numéro de la ligne à modifier.
LIST	Commande permettant de donner la liste, dans la fenêtre SBASIC, de tout ou partie du programme se trouvant en mémoire centrale.
NEW	Efface le programme se trouvant en mémoire centrale.
PLIST	Identique a la commande LIST, mais avec édition dans le tampon d'impression.
RENUM ou RENUMBER	Effectue la renumérotation totale ou partielle du programme en mémoire.
RUN	La commande RUN indique à SBASIC de démarrer l'exécution du programme se trouvant en mémoire centrale.
SAVE	Cette commande est utilisée pour stocker sur disque, sous forme source (sous forme texte), tout ou partie du programme SBASIC se trouvant en mémoire centrale.
STACK	Affiche la pile d'exécution du programme arrêté par une instruction STOP, un appui sur Ctrl-C ou la détection d'une erreur.
STEP	Relance, pour une seule ligne, l'exécution du programme arrêté par une instruction STOP ou un appui sur Ctrl-C.
SYSTEM ou EXIT	La commande SYSTEM est utilisée pour quitter SBASIC.
TRACE ON	Active la fonction TRACE.
TRACE OFF	Désactive la fonction TRACE.

Tableau 6 – Liste des commandes pures

4.7.2 Mots réservés ne pouvant pas être utilisés en mode direct

L'instruction READ et le mot réservé DATA ne peuvent s'utiliser qu'en mode programme. Un rapide coup d'œil à leur description détaillée en « Deuxième partie : Instructions et fonctions » permettra d'en comprendre la raison.

5 Utilisation des fichiers

Lorsqu'un programme SBASIC doit gérer des données en nombre trop important pour pouvoir être rentrées à chaque fois depuis le clavier, on utilise des fichiers sur disque que le programme peut lire lorsqu'il a besoin de ces données ou sur lesquels il peut écrire pour les mettre à jour.

Les fichiers que l'on peut utiliser sont de 2 types :

- Les Fichiers à accès séquentiel
- Les Fichiers à accès direct

Les premiers doivent s'écrire et se lire séquentiellement à l'instar d'une bande de magnétophone ou de magnétoscope alors que pour les seconds il est possible d'accéder directement à un enregistrement donné et de le modifier, ce qui correspond au fonctionnement d'un disque.

5.1 Fichier séquentiel d'entrée-sortie

La forme la plus simple d'utilisation des fichiers de données est la forme séquentielle dans laquelle les données sont écrites les unes derrière les autres, séparées par des Retour-Chariot ou par des virgules.

5.1.1 Ouverture en écriture d'un fichier séquentiel

Avant d'écrire dans un fichier sous SBASIC, il est nécessaire de « l'ouvrir ». C'est l'ordre OPEN qui est utilisé pour cette opération. Il permet d'affecter un canal au fichier concerné et de lui réserver en mémoire centrale une zone de **mémoire tampon d'entrées-sorties** ou simplement **tampon** (ou *buffer*), par laquelle transiteront les échanges avec le disque.

Imaginons que l'on veuille constituer un fichier contenant les 6 premiers entiers ainsi que leurs carrés avec quelques explications, à savoir :

- le libellé « Liste des 6 premiers entiers »,
- puis les 6 premiers entiers « 1 », « 2 », …, « 6 »,
- puis le libellé « leurs carrés »,
- enfin les 6 carrés « 1 », « 4 », …, « 36 ».

On commence par ouvrir un « nouveau » fichier au moyen de l'instruction :

```
100 OPEN NEW "CARRE" AS 1
```

qui indique que l'on veut ouvrir sur le canal 1 un fichier qui aura pour nom sur le disque, « CARRE », en fait « CARRE.DAT », car, par défaut, SBASIC utilise l'extension « .DAT » pour les fichiers de données.

Si un fichier de ce nom existait déjà sur le disque il est détruit sans avertissement, c'est en ce sens que tout fichier séquentiel ouvert en écriture est forcément un « nouveau » fichier.

5.1.2 Ecriture dans un fichier séquentiel

Pour écrire dans le fichier séquentiel précédemment ouvert sur le canal 1, il suffit d'utiliser une version légèrement modifiée de l'ordre PRINT.

La ligne :

```
110 PRINT #1 "Liste des 5 premiers entiers"
```

permet d'écrire, dans le fichier ouvert sur le canal 1, le libellé « Liste des 5 premiers entiers ».

De même la ligne :

```
120 FOR I%=1 TO 6: PRINT #1 I%: NEXT I%
```

permet d'y écrire les 6 premiers entiers « 1 », « 2 », …, « 6 ». Chacun étant séparé du précédent par un Retour-Chariot.

Pour terminer il suffira d'écrire :

```
130 PRINT #1 "leurs carrés"
140 FOR I%=1 TO 6: PRINT #1 I%*I%: NEXT I%
```

Toutefois il ne faut pas oublier de fermer le fichier par l'instruction :

```
200 CLOSE 1
```

Cette instruction permet tout d'abord de libérer le canal utilisé, ici le canal 1, afin de pouvoir éventuellement le réutiliser pour la lecture ou l'écriture d'un autre fichier.

Mais elle a une autre fonction bien plus importante dans l'exemple qui nous préoccupe : elle écrit effectivement sur disque les données qui peuvent encore se trouver dans le *buffer* associé au fichier. En effet, dans le but d'accélérer les opérations, l'écriture sur disque ne se produit pas effectivement lors de l'exécution de chaque ordre PRINT #1. Les données sont stockées dans un *buffer*, une mémoire tampon, dont la taille varie en fonction du système utilisé et qui est recopié sur le disque lorsqu'il est plein. C'est la seconde fonction de l'ordre CLOSE que d'effectuer l'écriture de cette partie résiduelle.

Après l'exécution de ce petit programme on doit trouver sur disque le fichier « CARRE.DAT » qui pourra être listé.

Le programme précédent peut paraître simpliste, mais il illustre la façon d'éviter de saisir à la main les éléments à écrire dans le fichier. Un programme plus général d'écriture en fichier doit comporter 2 phases bien distinctes :

- Une phase de *saisie* qui permettra à l'utilisateur, grâce à un certain nombre d'ordres INPUT, de rentrer des données en mémoire centrale en les frappant au clavier.
- Une phase d'écriture effective dans le fichier, qui grâce à des ordres PRINT, permettra de faire passer ces données de la mémoire centrale vers le fichier se trouvant sur le disque.

5.1.3 Ouverture d'un fichier en lecture

Après avoir exécuté le programme précédent et l'avoir, le cas échéant, sauvegardé, il est maintenant temps de savoir comment récupérer les données que l'on a écrites dans ce fichier, en vue de leur éventuelle utilisation par un programme SBASIC.

Il faut d'abord « ouvrir » le fichier pour le relire. C'est un fichier qui existe déjà sur le disque, un ancien fichier ; on utilisera donc l'instruction :

```
100 OPEN OLD "CARRE" AS 2
```

qui indique à SBASIC que l'on veut ouvrir le fichier « CARRE », en fait « CARRE.DAT », en vue de le lire.

5.1.4 Lecture d'un fichier séquentiel

De même que l'écriture dans un fichier séquentiel utilisait l'ordre PRINT #, la lecture utilise l'ordre symétrique : INPUT #.

On commencera par relire la première chaîne :

```
110 INPUT #2 A$
```

puis on pourra retrouver les 6 valeurs numériques que l'on avait écrites, mais il faut, pour ce faire, préparer un tableau NOMBRE pour les stocker en mémoire centrale. On écrira donc :

```
120 DIM NOMBRE(6): FOR I%=1 TO 6: INPUT #2 NOMBRE(I%): NEXT I%
```

Pour se convaincre que l'on a bien récupéré les valeurs précédemment écrites dans le fichier, on peut insérer dans cette boucle l'instruction PRINT NOMBRE(I%) ou même réaliser une autre boucle d'impression.

On récupère ensuite les autres valeurs de manière analogue :

```
130 INPUT #2 B$
140 DIM CARRE(6): FOR I%=1 TO 6: INPUT #2 CARRE(I%): NEXT I%
```

Enfin il faut fermer le fichier au moyen de l'instruction :

```
200 CLOSE 2
```

dont le seul effet, ici, est de libérer le canal utilisé.

Une première remarque s'impose à la lecture de ce programme : il faut connaître la structure logique du fichier pour être en mesure de le relire. Ici, il fallait savoir qu'il contenait un libellé, six valeurs numériques, un autre libellé et les six dernières valeurs numériques. Si l'on avait commis une erreur, par exemple en oubliant de lire le libellé central (omission de la ligne 130), une *erreur 30* se serait produite en ligne 140 puisque le programme aurait alors essayé de lire une valeur numérique et que le fichier lui aurait présenté le libellé central, constitué d'une chaîne de caractères.

Toutefois, chaque nombre étant stocké dans un fichier séquentiel sous forme de la chaîne de caractères qui permet de l'écrire, il est toujours possible de relire un fichier séquentiel en considérant tous ses éléments comme des chaînes de caractères.

Après avoir frappé et exécuté une première fois le programme précédent, l'utilisateur peut ajouter l'instruction :

```
150 INPUT #2 C$
```

et exécuter le programme ainsi modifié. Lors de l'exécution de la ligne 150, il se produit une *erreur 8*, (voir *« Annexe 1 : Liste des commandes, instructions et fonctions »)* qui indique la fin du fichier séquentiel. Cette erreur peut évidemment être gérée sous SBASIC, à condition de l'avoir prévu, en utilisant l'instruction ON ERROR GOTO.

Cette modification permet donc de se rendre compte qu'il n'est pas nécessaire de connaître le nombre exact d'éléments d'un fichier séquentiel. Il suffit dans ce cas de prévoir l'occurrence d'une erreur de fin de fichier dont la prise en compte permettra un branchement vers la suite du traitement.

5.1.5 Correspondance entre les ordres PRINT, PRINT #, INPUT, INPUT

Pour éviter toute confusion dans l'utilisation des instructions PRINT, PRINT #, INPUT, INPUT #, il est bon de se rappeler que SBASIC opère sur des données qui doivent se trouver en mémoire centrale.

Les instructions INPUT et INPUT # permettent d'introduire ces données en mémoire centrale, qu'elles proviennent du clavier ou d'un fichier disque qui, tous deux, sont des périphériques par rapport à la mémoire centrale. Le fichier fournit des données ASCII tout à fait comme si ces dernières étaient entrées au clavier.

De même, les ordre PRINT et PRINT # permettent de diriger des données vers les périphériques externes que sont l'écran et le disque. Dans le premier cas on écrit sur l'écran dans la fenêtre SBASIC, dans le second on écrit dans un fichier.

5.1.6 Autres séparateurs, cas particuliers

Jusqu'à présent nous avons, dans le fichier, séparé les données par des Retour-Chariot, mais il est aussi possible de les séparer par des virgules. Il suffit de mettre ces dernières dans la chaîne écrite dans le fichier à l'aide de l'ordre PRINT #.

Lorsqu'on utilise la virgule comme séparateur, il faut se souvenir qu'elle joue un rôle de séparateur aussi bien pour l'instruction INPUT que pour l'instruction INPUT #.

De même les limitations concernant les chaînes vides sont les mêmes que celles relatives à l'instruction INPUT. Donc si certains des éléments d'un fichier sont des chaînes contenant des virgules, voire des chaînes vides, il faudra utiliser l'instruction INPUT LINE # à la place de l'instruction INPUT #.

Il est aussi possible d'utiliser l'instruction PRINT # suivie du symbole « ; », ce qui a pour effet de ne pas introduire de Retour-Chariot après la donnée concernée par cet ordre. La donnée que l'on écrira ensuite dans le fichier lui sera alors concaténée et lors de la relecture il sera impossible de les dissocier par de simples ordres INPUT. Dans ce cas il ne faut pas oublier que la dernière donnée écrite dans le fichier doit être suivie d'un Retour-Chariot.

L'utilisation de PRINT # suivie du symbole « , » est autorisée. Elle introduit dans le fichier un nombre d'espaces correspondant à celui des espaces nécessaires à la tabulation liée à la virgule.

Enfin on peut aussi signaler l'instruction INCH$ qui permet de relire un fichier séquentiel caractère par caractère. Pour un fichier ouvert sur le canal 1, l'instruction :

```
R$=INCH$(1)
```

permettra d'affecter à la variable R$ le caractère suivant du fichier.

5.1.7 Utilisation du canal 0

5.1.7.1 Instruction INPUT sur le canal 0

Il est parfois souhaitable de saisir des données au clavier sans voir apparaître le signe « ? » à chaque exécution de l'ordre INPUT. Ceci est possible en utilisant le canal 0.

Exemple :

```
10 INPUT #0 B$
20 INPUT LINE #0 A$
```

Les instructions INPUT de cet exemple sont utilisés comme des ordres INPUT normaux.

5.1.7.2 Instruction PRINT sur le canal 0

Il est souvent nécessaire d'effectuer des sorties SBASIC sur un fichier plutôt que dans la fenêtre SBASIC. Le canal 0 permet d'aiguiller en sortie les ordres PRINT, vers la fenêtre SBASIC ou un fichier.

L'utilisation de l'instruction PRINT #0, sans celle préalable de l'instruction OPEN permet l'affichage direct de données dans la fenêtre SBASIC comme une instruction PRINT ordinaire.

Exemple :

```
20 PRINT #0 "CECI EST UN TEST"
```

affiche dans la fenêtre SBASIC « CECI EST UN TEST », comme si « #0 » n'était pas spécifié.

En revanche, l'ouverture du canal 0 permet d'aiguiller les données vers un fichier. L'utilisation de l'instruction OPEN NEW ou OPEN APPEND avec 0 comme numéro de canal 0 demande à SBASIC d'utiliser ce fichier de sortie chaque fois que l'ordre PRINT #0 est utilisé.

Exemple :

```
10 OPEN NEW "OUTPUT" AS 0
20 PRINT #0 "CECI EST UN TEST"
30 CLOSE 0
```

Le fichier « OUTPUT.DAT » est créé et contient le texte « CECI EST UN TEST ».

Les ordres PRINT ne spécifiant pas « #0 » continueront à afficher les données dans la fenêtre SBASIC L'ordre CLOSE 0 indiquera à SBASIC d'aiguiller à nouveau, vers la fenêtre SBASIC, toutes les données relatives à des instructions PRINT #0 et ce, jusqu'à l'exécution d'un autre ordre OPEN … AS 0.

L'exemple suivant illustre l'utilisation de l'instruction PRINT # sur le canal 0. Ce programme permet de choisir la fenêtre SBASIC ou un fichier pour éditer les résultats.

```
10 INPUT "ECRAN OU FICHIER ? (E / F)"; R$
20 IF R$ ="F" THEN OPEN NEW "OUTPUT" AS 0
30 FOR I=1 TO 10
40 PRINT #0 I, SQR(I)
50 NEXT I
60 IF R$="F" THEN CLOSE 0
70 END
```

5.2 Les tableaux virtuels

Le tableau virtuel est l'exemple le plus simple de fichier à accès direct. Il permet de spécifier qu'un ensemble de données doit résider sur disque plutôt qu'en mémoire centrale. A l'intérieur d'un programme SBASIC, les données virtuelles sont référencées comme des éléments de tableau, ce qui autorise puissance et simplicité d'emploi.

Les deux avantages de ce concept sont que le tableau peut avoir jusqu'à 2147483647 éléments, quelle que soit la taille mémoire centrale disponible, et que les données, dès leur affectation, résident sur le disque. Elles sont donc permanentes et peuvent être réutilisées.

La seule restriction au regard d'un fichier accédé par enregistrement, est que tous les éléments doivent être du même type : entiers, réels ou chaînes de caractères et que, dans ce dernier cas, toutes les chaînes doivent avoir la même longueur.

Le tableau virtuel permet d'accéder directement, en lecture et en écriture, à une donnée quel que soit son emplacement dans le fichier disque.

5.2.1 Ouverture d'un tableau virtuel

Avant d'être référencé, un tableau virtuel doit être ouvert au moyen de l'instruction OPEN, comme dans l'exemple suivant :

```
100 OPEN "VIRT" AS 1
```

Cette instruction va permettre d'accéder aux données du fichier « VIRT.DAT ». Mais pour indiquer qu'il s'agit d'un tableau virtuel et non d'un fichier accédé par enregistrement ainsi que pour en préciser le type, il faut dimensionner le tableau qui permettra de manipuler les éléments de ce fichier. On utilise pour cela une forme spéciale de l'instruction DIM.

Imaginons que l'on désire stocker les 10 premiers carrés « 1 », « 4 », …, « 100 ». On peut écrire :

```
110 DIM #1 NO%(10)
```

qui indique à SBASIC que le fichier ouvert sur le canal 1 est un tableau virtuel d'entiers pouvant contenir jusqu'à 11 éléments.

Pour ouvrir un fichier virtuel destiné à recevoir des chaînes de caractères, on utiliserait :

```
500 OPEN "TEXTE" AS 3
510 DIM #3 A$(50)=63
```

ce qui indiquerait à SBASIC que les données du fichier « TEXTE.DAT » sont les éléments du tableau A$(*) et que toutes ces chaînes ont une longueur de 63 caractères.

On retrouve ici l'une des contraintes signalées au début de ce chapitre : toutes les chaînes utilisées doivent avoir la même longueur pour que SBASIC puisse accéder directement à l'une d'entre elles. Dans l'exemple traité, elle est de 63 caractères.

Si aucune longueur n'est spécifiée, comme dans l'instruction :

```
510 DIM #3 A$(50)
```

la longueur prise par défaut est 18.

Si l'on affecte à l'un des éléments d'un tableau virtuel de chaînes de caractères une valeur de type alphanumérique dont la longueur est inférieure à celle des éléments du tableau, SBASIC complète ces valeurs par des espaces de façon à obtenir la longueur spécifiée. Dans le cas contraire, la valeur est tronquée.

Il est aussi possible de considérer les éléments d'un tableau virtuel comme ceux d'une **matrice** à deux dimensions. Si l'on veut, par exemple, traiter les éléments du fichier « MATRICE » comme ceux d'une matrice de nombre réels de 10 lignes et 5 colonnes, on écrira :

```
700 OPEN "MATRICE" AS 5
710 DIM #5 N(9,4)
```

Les tableaux virtuels à deux dimensions peuvent également contenir des entiers, des réels ou des chaînes de caractères ; dans ce dernier cas, il faut aussi spécifier la longueur des chaînes si l'on veut utiliser une longueur différente de la longueur par défaut.

SBASIC supporte un nombre quelconque de dimensions à condition que le nombre total d'éléments ne dépasse pas 2147483647.

5.2.2 Utilisation des tableaux virtuels

Après l'ouverture du fichier et la déclaration au moyen de l'instruction DIM, le tableau virtuel peut être utilisé dans des instructions d'affectation et dans des expressions comme un tableau ordinaire, avec, toutefois, les restrictions suivantes :

- Il ne peut apparaître comme paramètre ou comme élément de la liste d'appel d'un sous-programme (instruction SUB) ;

- Aucun de ses éléments ne peut être modifié au moyen des instructions SET, LSET, RSET et SWAP.

Pour créer le fichier contenant les 10 premiers carrés introduit plus haut, on écrit :

```
100 OPEN "VIRT" AS 1
110 DIM #1 NO%(10)
120 FOR I%=1 TO 10
130 NO%(I%)= I%* I%
140 NEXT I%
150 CLOSE 1
```

Dans ce programme, les écritures dans le fichier sont réalisées au moyen des affectations exécutées en ligne 130 ; il ne faut pas, pour un tableau virtuel, utiliser l'instruction PRINT # ; cette dernière est réservée aux fichiers séquentiels.

Toutefois on retrouve le même mécanisme que celui mis en œuvre lors de l'utilisation de fichiers à accès séquentiel : l'écriture effective dans le fichier n'est pas vraiment réalisée lors de l'exécution de la ligne 130. Les données sont seulement stockées dans une mémoire tampon (*buffer*) de transit qui est périodiquement écrite sur disque. Il faut donc se souvenir que l'instruction CLOSE de la ligne 150 permettra non seulement de libérer le canal utilisé par le tableau virtuel, mais qu'elle provoquera aussi l'écriture sur le fichier de la partie résiduelle des données qui pourrait se trouver encore dans la mémoire tampon.

Pour modifier l'une des valeurs du fichier d'entiers précédemment écrit sur disque sous le nom « VIRT.DAT », il suffit d'écrire :

```
100 OPEN "VIRT" AS 1
110 DIM #1, NO%(10)
120 INPUT "Rang de la valeur à modifier"; I%
130 IF I%=0 THEN 200
140 PRINT "Valeur courante :"; NO%(I%)
150 INPUT "Nouvelle valeur :"; NO%(I%)
160 GOTO 120
200 CLOSE 1: END
```

L'utilisation de la valeur NO%(I%), en ligne 140, suffit pour forcer SBASIC à la relire dans le fichier.

L'affectation réalisée grâce à l'instruction INPUT en ligne 150, suffit pour écrire la nouvelle valeur dans le fichier.

Un programme similaire de mise à jour peut aussi être écrit pour un fichier de chaînes.

5.2.3 Remarques sur les tableaux virtuels

Un fichier créé et référencé comme tableau virtuel ne comporte pas d'information sur la dimension du tableau et le type des données qu'il contient.

Un secteur de 512 octets, peut être considéré comme contenant :

- 64 réels (codés sur 8 octets chacun par la fonction CVTF$)
- 256 entiers (codés sur 2 octets chacun par la fonction CVT%$)
- 4 chaînes de 128 caractères
- 16 chaînes de 32 caractères

Une matrice est rangée par lignes, c'est-à-dire en commençant par la ligne 0 au début du fichier, suivie de la ligne 1, etc. Cette disposition est importante pour deux raisons :

- L'accès séquentiel aux éléments d'une matrice est plus rapide lorsqu'il est réalisé ligne par ligne que colonne par colonne. En effet l'accès par ligne réalisera une lecture séquentielle du fichier, secteur après secteur, alors que l'accès par colonne va multiplier le nombre de chargements en

mémoire des secteurs.

- La déclaration de dimension d'un tableau virtuel comprend un ou deux indices, alors qu'un fichier est une collection d'objets séquentiels qu'un programme peut exploiter en définissant, de façon externe au fichier, une structure plus élaborée -une matrice- qui n'est d'ailleurs pas forcément figée.

Supposons par exemple qu'un tableau virtuel ait été créé comme une matrice 50x50 par l'instruction :

```
DIM #1 A(49,49)
```

Le fichier disque correspondant contient alors 2500 éléments qui pourront par la suite être considérés comme une simple liste dont la dimension peut être inférieure à 2500, certains éléments restant ignorés du programme.

Ce même fichier pourra donc être ultérieurement ouvert en lecture et dimensionné au moyen de l'instruction

```
DIM #1 B(1000)
```

ce qui permettra d'accéder aux mille premiers éléments de ce fichier.

5.2.4 Extension d'un tableau virtuel

Lors de sa création, un fichier à accès direct ne contient aucun secteur accessible à l'utilisateur. La référence en lecture à un élément d'un tableau virtuel non écrit provoquera une *erreur 24* semblable à une erreur fin de fichier, en lecture d'un fichier séquentiel.

Un fichier à accès direct peut être étendu à tout moment en affectant une valeur à un élément du tableau virtuel extérieur au fichier. Chaque fois qu'une extension est demandée, SBASIC ajoute au fichier, si nécessaire, les enregistrements se trouvant dans le même secteur que 1'enregistrement écrit ; mais ces derniers sont alors remplis de caractères NUL (CHR$(0)).

Exemple :

```
100 OPEN "DATA" AS 1
120 DIM # 1 A(250)
130 PRINT A(250)
```

provoquera une *erreur 24*, car l'élément 250 n'existe pas dans le fichier nouvellement créé. Pour contenir 250 éléments le fichier doit être étendu de la façon suivante :

```
100 OPEN "DATA" AS 1
120 DIM # 1 A(250)
125 A(250)=0
130 PRINT A(250)
```

Nous avons décrit jusqu'à présent deux méthodes d'entrée-sortie disque : la méthode séquentielle avec les instructions PRINT # et INPUT #, et la méthode d'accès direct avec les tableaux virtuels. L'accès séquentiel est simple à utiliser, mais restreint les possibilités d'accès aux données contenues dans un fichier. Les tableaux virtuels assurent un accès sélectif rapide, mais ne permettent pas de mélanger différents types de données dans un même fichier.

SBASIC propose une troisième méthode d'entrée-sortie sur disque, la méthode d'accès direct par enregistrement. Les données sont accessibles de façon aléatoire et tous les types (entier, réel, chaîne de caractères) peuvent être présents dans le même fichier. Les fichiers utilisés dans ce cas sont aussi appelés **fichiers à accès direct**.

Cette méthode repose sur le fait que les données écrites sur disque sont contenues dans des enregistrements physiques de longueur fixe. Chaque enregistrement d'un fichier doit pouvoir être lu ou écrit sur demande, indépendamment des données qu'il contient. Dans chacun des enregistrements on peut mettre des chaînes de caractères aussi bien que des données numériques, entières ou réelles, écrites en binaire, ce qui évite leur conversion lors des opérations d'entrée-sortie. La méthode est donc efficace et rapide pour traiter les données contenues dans un fichier disque.

Dans la suite nous envisagerons le traitement d'un fichier du personnel, contenant pour chaque employé les renseignements suivants :

- Un nom contenant au plus 30 caractères,
- Un prénom contenant au plus 12 caractères,
- Un salaire, nombre réel codé sur 8 caractères par la fonction CVTF$.

On utilisera donc un fichier à accès direct dont chaque enregistrement aura pour longueur 50 octets, somme des longueurs de chacun des trois champs listés ci-dessus.

5.2.5 Ouverture des fichiers accessibles par enregistrement

Les fichiers auxquels on peut accéder par enregistrement sont des fichiers à accès direct semblables à ceux utilisés pour stocker les tableaux virtuels. Ils sont également ouverts au moyen l'instruction OPEN. Il faut noter que toute tentative d'accès direct sur un fichier séquentiel provoquera une erreur.

Si l'on veut créer, mettre à jour ou tout simplement utiliser le fichier à accès direct, « PERSON.DAT », décrit ci-dessus, contenant les noms, les prénoms et les salaires des employés d'une société, il suffit d'exécuter :

```
100 OPEN "PERSON" AS 1 LEN 50
```

qui indique a SBASIC de réserver le canal 1 pour communiquer avec le fichier disque « PERSON.DAT », dont chaque enregistrement a pour longueur 50.

5.2.6 Communication entre le disque et la mémoire centrale

Un fichier à accès direct communique avec la mémoire centrale de l'ordinateur par l'intermédiaire d'une zone tampon – **la mémoire tampon** - dont la longueur est égale à la longueur d'un enregistrement et qui contient l'enregistrement en cours de traitement.

Les instructions GET et PUT, que nous verrons dans la deuxième partie, permettent le transfert des données d'un enregistrement vers la mémoire tampon et inversement, de la mémoire tampon vers un enregistrement du disque. Il faut donc affecter des valeurs aux différents éléments de cette mémoire tampon ou, inversement, lire celles provenant du fichier.

Il convient donc au préalable définir les différentes zones de cette mémoire tampon en lui associant des variables. Cela est réalisé au moyen d'une forme spéciale de l'instruction FIELD ; chaque zone du tampon est alors identifiée par une variable de type chaîne de caractères.

Dans l'exemple qui nous préoccupe, nous utiliserons les variables suivantes :

- NOM$ d'au plus 30 octets pour le nom de l'employé
- PRE$ d'au plus 12 octets pour le prénom de l'employé
- SAL$ pour le salaire de l'employé, codé au moyen de la fonction CVTF$

Il faut alors associer ces variables à la mémoire tampon du fichier ouvert sur le canal 1, ce qui-est réalisé au moyen de l'instruction suivante :

```
110 FIELD #1, 30 AS NOM$, 12 AS PRE$, 8 AS SAL$
```

qui découpe le tampon d'entrée-sortie en trois zones associées respectivement aux variables NOM$, PRE$ et SAL$.

Mais on souhaite aussi désigner par NNN$ l'ensemble nom + prénom. Il suffit alors d'exécuter la ligne 115 après la ligne 110 :

```
115 FIELD #1, 42 AS NNN$
```

ce qui permettra de disposer de 4 variables de communication entre le programme et la mémoire tampon associée au fichier.

Il faut toutefois prendre garde au fait que les variables NOM$, PRE$, SAL$ et NNN$ sont d'un genre très particulier et qu'il ne faut jamais leur affecter de valeurs au moyen d'une instruction LET -

implicite ou explicite -, ce qui aurait pour conséquence d'annuler leur lien avec la mémoire tampon associée au fichier. Il ne faut donc utiliser que les instructions RSET, LSET ou SET lorsque l'on désire affecter directement une valeur à l'une de ces variables.

Il faut aussi se rappeler que le contenu de chacune d'entre elles est modifié chaque fois que l'on change le contenu de la mémoire tampon associée au moyen à l'instruction GET.

De même, il ne faut pas s'étonner que la modification du contenu de PRE$ par exemple, entraîne celle de la valeur de NNN$ dont PRE$ contient en fait les 12 derniers caractères.

On peut écrire dans la mémoire tampon grâce aux instructions suivantes :

```
200 RSET NOM$="DUPONT"
210 RSET PRE$="Jean"
```

ce qui donnera, à la variable NNN$, la valeur :

« DUPONT□□□□□□□□□□□□□□□□□□□□□□□□Jean□□□□□□□□□□□□ »

où le caractère « □ » représente un espace, les chaînes « DUPONT » et « Jean » ayant été complétées par suffisamment d'espaces pour s'adapter aux longueurs données pour les variables NOM$ et PRE$.

Pour enregistrer, par exemple, le salaire de 8715,25 dans la zone correspondante, il faut utiliser l'instruction :

```
215 RSET SAL$=CVTF$(8715.25)
```

qui écrit dans la troisième zone de la mémoire tampon la *représentation binaire* du nombre réel 8715.25. Cette représentation binaire n'a rien à voir avec la chaîne permettant d'imprimer ce nombre et qui serait obtenue au moyen de la fonction STR$; c'est une suite des huit octets permettant de coder ce nombre en mémoire centrale ; elle est illisible pour le commun des mortels, mais présente un double avantage :

- Elle offre un format fixe de 8 octets quelle que soit la valeur du nombre représenté,
- Elle évite les conversions lors de la lecture ou de l'écriture d'un enregistrement.

Si l'on veut enregistrer un nombre entier dans un fichier il faut lui associer, dans la mémoire tampon, une variable de longueur 2 et le coder grâce à la fonction CVT%$, qui permet d'obtenir la chaîne des deux octets représentant ce nombre en mémoire centrale.

5.2.7 Les instructions GET et PUT

L'écriture des données dans la mémoire tampon comme précédemment, il faut transférer le contenu de cette dernière vers le fichier. On réalise cela grâce à l'instruction PUT. Ainsi, l'instruction :

```
300 PUT #1 RECORD 5
```

permet d'écrire le contenu la mémoire tampon dans l'enregistrement numéro 5 du fichier :

DUPONT□□□□□□□□□□□□□□□□□□□□□□□□Jean□□□□□□□□8715.25

Si l'on exécute ensuite :

```
400 RSET PRE$="Jacques"
410 PUT #, RECORD 8
```

on transfère la mémoire tampon, qui vaut alors :

DUPONT□□□□□□□□□□□□□□□□□□□□□□□□Jacques□□□□□8715.25

dans l'enregistrement numéro 8 du fichier.

De même :

```
500 RSET NOM$="DURAND": RSET PRE$="Pierre": RSET SAL$=CVTF$(6123.20)
510 PUT #1 RECORD 12
```

permet d'écrire le contenu de la mémoire tampon :

DURAND□□□□□□□□□□□□□□□□□□□□□□□□□□□Pierre□□□□□□6132.20

dans l'enregistrement numéro 12 du fichier ouvert sur le canal 1.

Pour relire les données figurant dans un enregistrement du fichier, il faut commencer par les transférer dans la mémoire tampon au moyen de l'instruction GET, ce qui modifie automatiquement le contenu des variables associées à ce tampon.

Dans notre exemple, pour relire l'enregistrement numéro 8, il suffit d'exécuter :

```
600 GET #1 RECORD 8
```

Si l'on exécute ensuite :

```
610 PRINT NOM$, PRE$, CVT$F(SAL$)
```

on verra s'afficher dans la fenêtre SBASIC :

DUPONT□□□□□□□□□□□□□□□□□□□□□□□□□□□Jacques□□□□□8715.25

Il ne faut veiller à utiliser la fonction CVT$F, symétrique de la fonction CVTF$, qui permet de convertir en un nombre affichable la chaîne SAL$ codée en format interne.

Dans les instructions PUT et GET, la partie RECORD… est optionnelle : si elle n'est pas spécifiée, c'est l'enregistrement suivant le dernier auquel on a accédé qui est lu ou écrit (accès séquentiel).

Tenter de lire un enregistrement du fichier qui n'a pas été préalablement écrit provoque une erreur. Ecrire un enregistrement de numéro supérieur au dernier enregistrement écrit étend automatiquement le fichier.

Dans l'exemple qui nous intéresse, essayer de lire l'enregistrement numéro 17, provoque une *erreur 24*.

Il en est de même pour l'accès à l'enregistrement numéro 5.

5.2.8 Extension des fichiers à accès direct

Le mécanisme de création et d'extension des fichiers à accès direct est le même que pour les fichiers associés aux tableaux virtuels. Lors de sa création, le fichier ne contient aucun enregistrement. Tenter de lire un enregistrement inexistant provoque une *erreur 24*. Le fichier peut être étendu à tout moment par l'écriture d'un enregistrement qui n'existe pas encore.

Chaque fois qu'un enregistrement est écrit, SBASIC ajoute au fichier, si nécessaire, les enregistrements se trouvant dans le même secteur et qui n'ont pas encore été écrits ; ils sont alors remplis de *caractères nuls*, tels que ceux obtenus au moyen de la fonction CHR$(0).

Par exemple, l'exécution du programme :

```
10 KILL "TESTD"
20 OPEN "TESTD" AS 1
30 FIELD #1, 100 AS G$
40 GET #1 RECORD 20
```

provoque une *erreur 24*. Il est donc nécessaire d'étendre le fichier avant toute lecture de l'enregistrement numéro 20 :

```
10 KILL "TESTD"
20 OPEN "TESTD" AS 1
30 FIELD #1, 100 AS G$
34 LSET G$="ENREGISTREMENT NUMERO 20"
38 PUT #1 RECORD 20
40 GET #1 RECORD 20
```

5.2.9 Exemple d'utilisation d'un fichier à accès direct

Le programme suivant montre comment on peut accéder à un fichier à accès direct. On considère le

fichier du personnel précédemment décrit, dont chaque enregistrement contient des renseignements sur un employé : son nom, son prénom, son salaire. Le numéro d'enregistrement identifie l'employé. Le programme imprime les éléments correspondant à un employé dont on a donné le numéro d'identification et en permet la modification.

Le nom de l'employé est rangé dans les 30 premiers caractères de l'enregistrement, son prénom dans les 12 suivants et son salaire dans les 8 derniers.

```
98 REM *** Ouverture du fichier et définition des variables ***
100 OPEN "PERSON" AS 1 LEN 50
120 FIELD #1, 30 AS NOM$, 12 AS PRE$, 8 AS SAL$
125 FIELD #1, 42 AS NNN$

198 REM *** Saisie du numéro d'enregistrement à modifier ***
199 ON ERROR GOTO 250
200 INPUT "NUMERO DE L'EMPLOYE"; I%
210 IF I%=0 THEN 500
220 GET #1 RECORD I%
230 PRINT "Employé :"; NNN$
240 PRINT "Salaire : "; CVT$F(SAL$) : GOTO 300

248 REM *** Gestion de l'erreur "Enregistrement non trouvé"***
250 PRINT "Enregistrement inexistant"
260 RESUME 200

298 REM *** Mise à jour du contenu de la mémoire tampon (buffer)***
300 PRINT "Nom " ; : INPUT LINE R$
305 IF R$ <> "" THEN LSET NOM$=R$
320 PRINT "Prénom " ; : INPUT LINE R$
325 IF R$<>"" THEN LSET PRE$=R$
330 PRINT "Salaire " ; : INPUT LINE R$
335 IF R$<>"" THEN LSET SAL$=CVTF$(VAL(R$))
398 REM *** Ecriture de l'enregistrement ***
400 PUT #1 RECORD I%: GOTO 200
498 REM *** Fin de programme
500 CLOSE 1
510 END
```

Le fichier « PERSON.DAT » est ouvert sur le canal 1, puis les variables NOM$, PRE$, SAL$ et NNN$ sont associées à la mémoire tampon (*buffer*).

La ligne 200 demande le numéro identifiant l'employé afin de pour lire l'enregistrement le concernant. La valeur 0 permet de terminer la session. En cas de demande par l'utilisateur d'un enregistrement non présent, la ligne 210 permet un branchement en ligne 250 qui affiche un message explicatif. Dans les autres cas, les caractéristiques de l'employé sont affichées.

Les lignes 300 à 399 permettent la modification des données écrites dans le fichier. L'utilisation de l'instruction INPUT LINE autorise une réponse vide et, dans ce cas, de ne pas modifier la valeur affichée.

La ligne 400 provoque la réécriture dans le fichier du contenu la mémoire tampon (*buffer*) puis boucle sur une nouvelle demande.

Enfin la ligne 500 permet de fermer le fichier et termine le programme.

6 Gestion des erreurs

Les erreurs qui peuvent être signalées au cours de l'exécution d'un programme se répartissent en deux classes :

- Les erreurs **d'entrée-sortie** (disque, clavier, écran).

- Les erreurs de **syntaxe** ou **d'exécution**.

La liste complète des numéros d'erreur et leur signification est donnée en 0 « Liste des codes d'erreurs ». Les numéros d'erreur sont compris entre 1 et 49 pour les erreurs d'entrée-sortie et sont supérieurs à 49 pour les erreurs syntaxe ou d'exécution.

En général SBASIC signale l'erreur et arrête l'exécution du programme. Il est toutefois parfois utile de pouvoir continuer l'exécution en particulier quand les erreurs sont du premier type. L'ordre ON ERROR permet de gérer les erreurs signalées par SBASIC.

Il faut noter que la gestion des erreurs est locale aux sous-programmes appelés par l'instruction CALL.

6.1 Activation d'une procédure de gestion d'erreur

L'instruction ON ERROR GO TO *num-ligne*, où *num-ligne* désigne un numéro de ligne, strictement positif, permet de gérer les erreurs détectées par SBASIC. *num-ligne* est le numéro de la ligne où doit se brancher le programme en cas d'erreur.

Chaque fois que SBASIC détecte une erreur, il vérifie si une instruction ON ERROR GOTO *num-ligne* a été préalablement exécutée. Si tel est le cas, le contrôle est transféré à la ligne *num-ligne* indiquée dans l'instruction ON ERROR GOTO.

La syntaxe d'une ligne de programme utilisant cette instruction est la suivante :

num-ligne-1 ON ERROR GO TO *num-ligne-2*

Pour être efficace cette instruction doit être exécutée avant toute instruction susceptible de provoquer une erreur.

Dans sa **procédure de gestion d'erreur** l'utilisateur peut tester et utiliser les variables systèmes ERR et ERL qui contiennent, respectivement, le numéro de la dernière erreur rencontrée et le numéro de la ligne où elle s'est produite.

6.2 Reprise après gestion d'erreur

L'instruction RESUME permet de reprendre l'exécution du programme principal une fois le programme de gestion d'erreur exécuté. Elle permet de réactiver la procédure de gestion d'erreur associée à la dernière instruction ON ERROR GOTO *num-ligne* exécutée.

La syntaxe en est :

num-ligne RESUME [*num-ligne*]

Si un numéro de ligne est fourni après RESUME, SBASIC reprendra l'exécution du programme à cette ligne. Si aucun numéro de ligne n'est fourni (ou si ce numéro est 0) SBASIC interprète cette instruction en réexécutant toute la ligne où l'erreur s'est produite, **et non pas uniquement l'instruction qui a provoqué l'erreur**.

Il ne faut pas oublier que le rôle de cette instruction ne se limite pas à celui d'un branchement, que nous venons de mettre en évidence. Elle réactive aussi la procédure de gestion d'erreur. Sa substitution par une simple instruction GOTO ne permettrait pas de prendre en charge l'erreur qui pourrait se produire dans la suite du programme.

6.3 Inhibition d'une procédure de gestion d'erreur

Parfois, seules certaines sections de programmes nécessitent l'activation d'une procédure de gestion d'erreur. Il est possible d'annuler l'ordre ON ERROR GOTO *num-ligne* en utilisant l'une ou l'autre des lignes :

num-ligne ON ERROR GOTO

num-ligne ON ERROR GOTO 0

Ces deux lignes sont équivalentes et ont pour effet d'annuler tous les ordres ON ERROR GOTO activés.

Si l'on se trouve en situation de récupération d'erreur et que l'on n'a pas encore exécuté d'instruction RESUME, cette instruction annule la prise en charge de l'erreur et rend la main à SBASIC qui affiche l'erreur dans le format standard puis arrête l'exécution du programme.

Il faut aussi se souvenir que l'exécution d'une instruction ON ERROR GOTO à l'intérieur d'un sous-programme permet d'activer **localement** la procédure de gestion d'erreur référencée dans le numéro de ligne.

Si une procédure de gestion d'erreur était active lors de l'appel du sous-programme, le numéro de ligne correspondant est mémorisé. Cette procédure est réactivée lors de l'exécution de l'instruction RETURN terminant le sous-programme.

6.4 Exemples de gestion d'erreurs

Dans le programme suivant, l'utilisateur doit entrer 3 nombres dont le programme édite la somme dans la fenêtre SBASIC.

```
100 ON ERROR GOTO 1000
110 INPUT "ENTREZ 3 NOMBRES", A, B, C
120 PRINT "LA SOMME EST "; A+B+C
140 GOTO 110

1000 IF ERR<>30 THEN GOTO 1030
1010 PRINT " ENTREZ UNIQUEMENT DES NOMBRES"
1020 RESUME
1030 ON ERROR GOTO
```

Ce programme prévoit et prend en charge une possible erreur de l'utilisateur. Ce dernier pourrait en effet entrer des caractères non numériques, provoquant ainsi une *erreur 30* lors de l'addition des variables correspondantes, à la ligne 120. Le programme gérant cette erreur débute en ligne 1000, mais il ne récupère que l'*erreur 30*. Les autres provoqueront l'arrêt du programme : c'est, entre autres, le cas de *l'erreur 34*, associée à l'utilisation des touches Ctrl-C, qui permet à l'utilisateur d'interrompre le programme.

Le programme suivant demande de saisir le nom d'un fichier. Si ce dernier est présent sur le disque, travail le programme liste son contenu dans la fenêtre SBASIC.

```
10 INPUT "TAPEZ LE NOM DU FICHIER AVEC SON EXTENSION"; F$
20 ON ERROR GOTO 1000
30 OPEN OLD F$ AS 1
40 INPUT LINE #1 L$
50 PRINT L$
60 GOTO 40
100 CLOSE 1
110 END
1000 IF ERR = 8 THEN PRINT "Fin de fichier": END
1010 IF ERR = 4 THEN PRINT "Fichier absent": END
1020 PRINT "Erreur";ERR;"à la ligne";ERL: RESUME 10
```

Si Le fichier n'existe pas (*Erreur système numéro 4*), le programme affiche le message « Fichier absent » puis s'arrête.

Si le fichier existe sur le disque, le programme le liste, puis affiche le message « Fin de fichier » lorsque la fin du fichier est rencontrée (*Erreur système 8*), puis il s'arrête.

Pour les autres erreurs, par exemple si le nom du fichier est incorrect (*Erreur système 21*), il affiche

« Erreur *numéro de l'erreur* à la ligne *numéro de ligne* » et reprend l'exécution à la ligne 10.

7 Gestion des programmes volumineux

7.1 L'instruction CHAIN

Lorsqu'un programme est trop grand pour tenir en mémoire et y être exécuté, il doit être divisé en plusieurs segments. L'instruction CHAIN permet de charger et d'exécuter successivement plusieurs programmes. Les segments de programme doivent être soit de type « .BAS », soit des programmes compilés de type « .BAC ». Chaque segment est un fichier et n'importe quel programme résidant sur disque peut être appelé par CHAIN. La syntaxe en est :

num-ligne CHAIN *nom-fich* [*exp-num*]

Le fichier référencé par *nom-fich* est chargé. L'expression *exp-num*, optionnelle[3], désigne le numéro de ligne où le programme nouvellement chargé doit commencer à être exécuté. Notez l'absence de virgule entre les deux expressions.

Exemple :

```
1300 CHAIN "BALANCEZ" 100
```

Cette ligne charge le fichier BALANCEZ.BAC et lance son exécution à la ligne 100.

Chaîner un programme **compilé** (de type .BAC), est plus efficace que chaîner un programme écrit en langage source (.BAS). Les communications entre programmes ainsi chaînés doivent être réalisés par l'intermédiaire de fichiers disque. En effet, lorsque l'ordre CHAIN est exécuté, tous les fichiers qui étaient ouverts sont fermés, le nouveau programme est chargé, les anciennes variables sont détruites, et l'exécution du nouveau programme commence. Les fichiers utilisés conjointement par plusieurs programmes doivent être ouverts dans chacun d'entre eux et les variables réinitialisées.

7.2 L'instruction OVERLAY

Dans les versions anciennes de SBASIC, l'instruction OVERLAY permettait de réserver une zone fixe de la mémoire pour y placer les lignes de programme. Elle divisait la mémoire disponible en une zone « programme » et une zone « données ». Ainsi, l'insertion de nouvelles lignes de programme n'affectait pas les données qui étaient alors protégées.

Les zones « programme » et « données » sont maintenant disjointes. L'instruction OVERLAY est sans effet. Elle est cependant conservée pour assurer la compatibilité avec les anciens programmes.

La syntaxe en est :

num-ligne OVERLAY *nbre-ligne*

où *nbre-ligne* est un nombre inférieur ou égal à 511.

Avec SBASIC, il est possible de charger des sous-programmes dans la zone « programme » en mode programme avec les instructions LOAD et BLOAD.

Par exemple, le programme suivant :

```
10 OVERLAY 50
99 REM *** saisie des éléments ***
100 INPUT "Nombre d'éléments "; NE% : DIM EL(NE%)
110 FOR X% = 1 TO NE%
120 PRINT "élément "; X%;
130 INPUT EL(X%)
140 NEXT X%
199 REM *** Choix de la fonction ***
200 PRINT "0 - QUITTER          1 - Autre tableau"
```

[3] Dans la description d'une instruction, les éléments optionnels sont placés entre crochets.

```
210 PRINT "2 - Somme   3 - Moyenne     4 - Ecart-type"
220 I%=INSTR(1,"01234",INCH$(0))
230 IF I% = 0 THEN PRINT CHR$(7);: GOTO 220
250 ON I% GOTO 300, 310, 320, 330, 340
300 END
305 REM *** Traitement de la fonction demandée ***
310 CLEAR EL(*): GOTO 100
320 BLOAD "SOMME" : CALL SOMME(NE%,EL(*)) : GOTO 200
330 BLOAD "MOYEN" : CALL MOYEN(NE%,EL(*)) : GOTO 200
340 BLOAD "ECART" : CALL ECART(NE%,EL(*)) : GOTO 200
```

Programme SOMME.BAC :

```
1000 SUB SOMME(N%, E(*))
1010 LOCAL I%,X
1020 FOR I%=1 TO N%
1030 X=X+E(I%)
1040 NEXT I%
1050 PRINT "Somme =";X
1060 RETURN
```

Programme MOYEN.BAC :

```
1000 SUB MOYEN(N%, E(*))
1010 LOCAL I%,X
1020 FOR I%=1 TO N%
1030 X=X+E(I%)
1040 NEXT I%
1050 PRINT "Moyenne =";X/N%
1060 RETURN
```

Programme ECART.BAC :

```
1000 SUB ECART(N%, E(*))
1010 LOCAL I%,X,Y
1020 FOR I%=1 TO N%
1030 X=X+E(I%)
1040 NEXT I%
1050 X=X/N%
1060 FOR I%=1 TO N%
1070 Y=Y+(E(I%)-X)**2
1080 NEXT I%
1090 PRINT "Ecart =";SQR(Y/N%)
1100 RETURN
```

Ce programme commence par l'instruction OVERLAY pour réserver une zone de 50 lignes. Cette instruction est ignorée et peut être supprimée. Puis l'utilisateur est invité à saisir une série de données numériques (lignes 110 à 140). Il choisit ensuite le type de calcul (somme, moyenne ou écart-type) devant être effectué sur cette série. Une même zone du programme peut donc être utilisée pour des traitements différents, en fonction des besoins.

Cet exemple simple permet de se rendre compte de toute la puissance du SBASIC.

8 Gestion de l'imprimante

Sous Windows, les impressions ne se font plus en direct. C'est pourquoi SBASIC gère un tampon d'impression. Les commandes du sous-menu « Imprimante » de la fenêtre SBASIC permettent de vider le tampon d'impression ou d'éditer son contenu et l'imprimer à travers NOTEPAD.

Deuxième partie : Instructions et fonctions

1 Syntaxe et abréviations utilisées

- Les mots réservés du SBASIC, à savoir les noms des instructions, fonction et commandes, ainsi que les mots-clés qui y apparaissent sont écrits en lettres majuscules.
- Les paramètres associés aux instructions ou fonctions sont représentés par des mnémoniques en *italique*.

  Exemple :

  INT(*exp-num*)
- Lorsqu'une instruction fait appel à plusieurs paramètres de même type, les mnémoniques sont suivies d'un numéro d'ordre.

  Exemple :

  CURSOR *exp-num-1*,*exp-num-2*
- Les paramètres optionnels sont placés entre crochets.

  Exemple :

  DELETE [*num-ligne-1*] [,*num-ligne-2*]
- Les points de suspension indiquent que les paramètres précédents peuvent être répétés.

  Exemple :

  DATA *const-1*[,*const-2*] …
- Les éléments s'excluant mutuellement sont placés entre accolades et séparés par un trait vertical. Dans l'exemple ci-dessous, l'une des 2 ponctuations, virgule **ou** point-virgule, doit être marquée, mais pas les deux :

  { , | ; }
- Sauf spécification contraire, dans une liste de paramètres optionnels séparés par des virgules, l'emplacement de chaque paramètre omis doit être marqué par une virgule. Dans l'exemple ci-dessous, le premier paramètre de la commande est omis.

  LIST ,60
- La représentation de certains caractères particuliers est la suivante :

Touche retour chariot (↵)	Retour-Chariot
Appui simultané sur les touches Ctrl et X	Ctrl-X (X étant une touche quelconque du clavier)
Caractère Espace, lorsqu'il doit être mis en évidence	□

- Les mnémoniques utilisées dans les instructions ou fonctions et leur signification sont décrites ci-dessous :

Mnémonique	Signification
Adresse	Adresse (numéro de ligne ou étiquette)
Chaîne	Chaîne de caractères
Commande	Commande système
Const	Constante
Etiquette	Nom d'étiquette
Exp	Expression
exp-chaîne	Expression chaîne de caractères
exp-entière	Expression numérique entière
exp-logique	Expression logique
exp-num	Expression numérique
exp-réelle	Expression numérique réelle
instr-comp	Instruction composée
nom-fich	Nom de fichier
num-ligne	Numéro de ligne
séparateur	Séparateur (« ; » ou « , »)
nom-sprog	Nom de sous-programme
var	Variable
var-chaîne	Variable chaîne de caractères
var-num	Variable numérique

Tableau 7 - Mnémoniques utilisées dans les instructions

- Dans les descriptions qui suivent, on trouvera pour chaque ordre SBASIC :
  - Son type : instruction, fonction ou commande
  - Sa description succincte ainsi que celle de ses paramètres,
  - Un ou plusieurs exemples,
  - Une description détaillée,
  - Le ou les noms des instructions, fonctions ou commandes qui y sont liés.

2 Description détaillée

ABS(*exp-num*)	Fonction

Donne la valeur absolue d'une expression numérique.

exp-num Expression numérique quelconque.

Exemple

```
10 FOR X = -5 TO 5
20 PRINT X, ABS(X)
30 NEXT X
```

A l'exécution, le programme affiche :

```
-5      5
-4      4
-3      3
-2      2
-1      1
 0      0
 1      1
 2      2
 3      3
 4      4
 5      5
```

Action

La fonction ABS retourne la valeur absolue de *exp-num*.

Si *exp-num* est positive, la valeur retournée est celle de *exp-num* ; si elle est négative, la valeur retournée est l'opposé de *exp-num*.

Voir aussi : SGN.

ARGC	Variable système

Variable système contenant le nombre d'arguments de la ligne de commande SBASIC.

Exemple

```
10 PRINT "Nombre d'arguments ="; ARGC
20 IF ARGC = 0 THEN 60
30 FOR I=0 TO ARGC-1
40 PRINT "Argument";I;" : ";ARGV$(I)
50 NEXT I
60 END
```

Ce programme affiche le nombre d'arguments et les arguments de la ligne de commande.

Action

ARGC est une variable système (de même que ARGV$, XPEN, YPEN, ERR, ERL, PI ou DATE$).

Elle donne le nombre d'arguments présents sur la ligne de commande utilisée lors de l'invocation de

SBASIC.

La variable ARGC peut être utilisée dans des expressions comme toute autre variable numérique, mais elle ne peut pas figurer à gauche du signe égal (=) dans un ordre d'affectation (LET explicite ou implicite).

Voir aussi : ARGV$.

ARGV$(exp-num)	Variable système

Variable système contenant les arguments de la ligne de commande SBASIC.

exp-num Indice argument. .

Exemple (fichier ARG.BAS)

```
10 PRINT "Nombre d'arguments ="; ARGC
20 IF ARGC = 0 THEN 60
30 FOR I=0 TO ARGC-1
40 PRINT "Argument";I;" : ";ARGV$(I)
50 NEXT I
60 END
```

Ce programme affiche le nombre d'arguments et les arguments de la ligne de commande.

Si la ligne de commande est:

```
SBASIC ARG.BAS ARG1 "ARG 2" 3
```

il affiche:

```
Nombre d'arguments = 4
Argument 0 : TEST.BAS
Argument 1 : ARG1
Argument 2 : ARG 2
Argument 3 : 3
```

Action

ARGV$ est une variable système (de même que ARGC, YPEN, XPEN, YPEN, ERR, ERL, PI ou DATE$).

C'est un tableau de chaines de caractères de dimension ARGC. Chaque élément est un argument de la ligne de commande utilisée lors de l'invocation de SBASIC. Le premier argument (indice 0) est le nom du fichier SBASIC.

ARGV$(exp_num) ne peut pas figurer à gauche du signe égal « = » dans un ordre d'affectation de données (LET explicite ou implicite).

Voir aussi : ARGC.

ASC(*exp-chaîne*)	Fonction

Donne le code ASCII du premier caractère d'une chaîne de caractères.

exp-chaîne Expression chaîne de caractères.

Exemple 1

```
10 A% = ASC("1")
20 PRINT A%
```

Après exécution de ces lignes, la variable A% contient la valeur du code ASCII du caractère « 1 », à savoir 49.

Exemple 2

```
10 A$ = "ZORRO"
20 PRINT ASC(A$)
```

Ce programme affiche la valeur 90, correspondant au code ASCII de la lettre Z.

Action

La fonction ASC renvoie le code ASCII du premier caractère de *exp-chaîne*.

Si *exp-chaîne* contient plus d'un caractère, seul le code ASCII du premier caractère est renvoyé ; si *exp-chaîne* ne contient aucun caractère, la valeur 0 est retournée.

Voir aussi : CHR$.

ATN(*exp-num*) — Fonction

Retourne l'arc tangente, en radians, d'une expression numérique.

exp-num	Expression numérique quelconque.

Exemple

```
10 PRINT PI - 4*ATN(1)
```

La valeur retournée est égale à zéro, puisque l'arc tangente de 1 correspond à PI/4.

Action

La valeur retournée est comprise entre -PI/2 et PI/2, où PI/2 est approximativement égal à 1.5707963267948966.

Voir aussi : TAN.

BLOAD *nom-fich* [,,*num-ligne*] — Instruction

Charge en mémoire un programme sauvegardé sous forme **compilée**, avec une translation éventuelle des numéros de lignes.

nom-fich	Nom du programme à charger.
,,	Séparateurs.
num-ligne	Translation.

Exemple 1

```
10 BLOAD "ESSAI"
```

Le programme « ESSAI.BAC » est chargé, sans modification de sa numérotation.

Exemple 2

```
10 BLOAD "ESSAI",,100
```

Le programme « ESSAI.BAC » est chargé avec une translation de 100 dans la numérotation (chaque numéro de ligne est augmenté de 100).

Action

L'instruction BLOAD charge en mémoire le programme compilé *nom-fich*, en ajoutant éventuellement *num-ligne* à chaque numéro de ligne chargée.

nom-fich doit avoir été compilé préalablement au moyen de la commande COMPILE ; son extension par défaut est « BAC ».

Si *num-ligne* est omise, le programme chargé conserve la numérotation initiale. *num-ligne* ne peut être spécifié que si le programme a été compilé avec au moins une option C, S, O ou L ou un numéro de ligne en paramètre.

Les virgules « ,, » assurent la conformité syntaxique avec les commandes LOAD et SAVE.

BLOAD s'utilise en mode programme ou en mode direct.

En mode programme, BLOAD n'est autorisé que si le programme a été compilé avec au moins une option C, S, O ou L ou un numéro de ligne en paramètre. Lors de son chargement, les lignes de ce programme s'ajoutent aux lignes déjà présentes. Seules les lignes dont le numéro se trouve entre la première ligne et la dernière ligne du programme compilé sont effacées. Une erreur 89 est signalée si une de ces lignes est référencée (ligne courante ou CALL, GOSUB ou FOR en cours).

En mode direct, si le programme n'a pas été compilé avec une option C, S, O ou L ou un numéro de ligne en paramètre, le programme se retrouve, une fois rechargé, dans son état d'origine. Toutes les lignes déjà présentes sont effacées.

Voir aussi : LOAD, COMPILE.

BLOAD *nom-fich var-chaîne* — Instruction

Charge dans une variable chaîne le contenu d'un fichier.

nom-fich	Nom du fichier dont le contenu est à charger.
var-chaîne	Variable chaîne recevant le contenu du fichier.

Exemple

```
10 DIM B$(5)
20 FOR I% = 0 TO 4
30 BLOAD "BOUTON"+LTRIM$(RTRIM$(STR$(I%)))+".BMP" B$(I%)
40 NEXT I%
```

Le contenu des fichiers BOUTON0.BMP, BOUTON1.BMP, BOUTON2.BMP, BOUTON3.BMP et BOUTON4.BMP est chargé dans chacun des éléments du tableau de chaînes B$. Si ces fichiers contiennent des images au format bitmap, celles-ci peuvent être affichées avec l'instruction GPUT.

Action

L'instruction BLOAD charge dans une variable chaîne le contenu complet d'un fichier.

Voir aussi : BSAVE, GPUT.

BROFF ou BREAK OFF — Instruction

Inhibe l'arrêt d'un programme en cours d'exécution par appui sur les touches Ctrl-C.

Exemple

```
BROFF
```

Après l'exécution de cette commande, il n'est plus possible d'arrêter l'exécution du programme au

moyen de Ctrl-C.

Action

L'instruction BROFF (BREAK OFF) inhibe l'action des touches Ctrl-C : il n'est alors plus possible d'arrêter un programme en cours d'exécution en tapant Ctrl-C.

La notation alternative BREAK OFF n'est utilisable qu'en mode direct.

Voir aussi : BRON.

BRON ou BREAK ON — Instruction

Autorise l'arrêt d'un programme en cours d'exécution par appui sur Ctrl-C.

Exemple

```
BRON
```

Après l'exécution de cette instruction, le programme peut être arrêté en cours d'exécution par une pression simultanée des touches Ctrl et C.

Action

La commande BRON (BREAK ON) valide la possibilité d'arrêter l'exécution d'un programme par appui sur Ctrl-C. Cette option est prise par défaut lors du chargement de SBASIC.

Après l'arrêt, le programme n'est pas altéré. L'exécution peut reprendre en tapant CONT. Si le programme a été arrêté sur une instruction INPUT, la reprise s'effectue au début de la ligne contenant l'instruction INPUT.

La notation alternative BREAK ON n'est utilisable qu'en mode direct.

Voir aussi : BROFF.

BSAVE *nom-fich var-chaîne* — Instruction

Copie le contenu d'une variable chaîne dans un fichier.

nom-fich	Nom du fichier à écrire.
var-chaîne	Variable chaîne dont le contenu est écrit dans le fichier.

Exemple

```
10 GGET B$,100,50,109,59
20 BSAVE "ICONE.BMP" B$
```

Ce programme crée le fichier ICONE.BMP contenant l'image bitmap du rectangle de 10 x 10 pixels dont l'angle inférieur gauche se trouve à l'abscisse 100 et l'ordonnée 50 de la fenêtre graphique.

Action

L'instruction BLOAD charge dans une variable chaîne le contenu complet d'un fichier.

Voir aussi : BLOAD, GGET.

CALL *étiquette* [(*param-1, param-2, …*)] — Instruction

Permet d'appeler un sous-programme défini par une instruction SUB.

étiquette	Nom du sous-programme.
param-n	Expression, variable, élément de tableau ou tableau.

Exemple

```
110 INPUT "Nombre d'éléments ";NE%
120 DIM BB(NE%)
130 FOR X% = 1 TO NE%
140 PRINT "élément ";X%,
150 INPUT BB(X%)
160 NEXT X%
170 CALL SOMME (BB(*),NE%,SOM)
180 PRINT "La somme des ";NE%;" éléments introduits est égale à : ";SOM
190 END
…
210 SUB SOMME (A(*),N%,S)
220 S = 0
230 FOR X% = 1 TO N%
240 S = S + A(X%)
250 NEXT X%
260 RETURN
```

Ce programme fait appel au sous-programme SOMME spécifié dans l'instruction SUB. Les variables passées (BB(*), NE% et SOM) sont du même type que les paramètres formels associés à la définition du sous-programme, mais ne portent pas nécessairement le même nom.

Action

L'instruction CALL permet d'appeler un sous-programme défini au moyen d'une instruction SUB, en indiquant son nom ainsi que, le cas échéant, la liste des paramètres appel. Cette instruction est spécifique au SBASIC.

(*param-1*, *param-2* …) est la liste des paramètres d'appels. Cette liste doit contenir le même nombre d'éléments que la liste de paramètres formels présente dans l'instruction SUB.

Un paramètre d'appel peut-être une expression, une variable, un élément de tableau ou un tableau, à l'exclusion des tableaux virtuels ou des éléments de tableaux virtuels.

Chaque paramètre d'appel correspond au paramètre formel de même rang de l'instruction SUB appelée et doit être de même type. Les paramètres d'appel (ou *arguments*) doivent être identiques en type (entier, réel ou chaîne de caractères), et de même nature (variable ou tableau). A défaut, une erreur 99 est signalée. Toutefois, il est possible de remplacer une variable par un élément d'un tableau.

Un paramètre formel et le paramètre d'appel correspondant ne portent pas nécessairement le même nom.

Voir aussi : SUB, RETURN.

CALL *#exp-num exp-chaîne* (*var-1*, *var-2*, …) — Instruction

Permet d'appeler une fonction d'une librairie ouverte par OPEN LIBRARY.

exp-num	Numéro de canal compris entre 1 et 32.
exp-chaîne	Chaîne de caractères contenant le nom de la fonction.
var-n	Variable quelconque.

Exemple

```
10 OPEN LIBRARY "MaDll" AS 10
20 CALL #10 "MaFonction"(X%)
30 PRINT X%
40 CLOSE 10
```

Ce programme fait appel à la fonction « MaFonction » de la librairie « MaDll.DLL » L'adresse de la variable X% est attendue par la fonction. La fonction peut récupérer et/ou modifier le contenu de cette variable.

Action

L'instruction CALL# permet d'appeler une fonction d'une librairie à chargement dynamique (DLL) ouverte par OPEN LIBRARY en indiquant son nom dans une chaine de caractères ainsi que la liste des variables dont les adresses sont passées en paramètres. Cette instruction est spécifique au SBASIC.

(*var-1*, *var-2* …) est la liste des variables dont les adresses sont passées en paramètre. Le type et le nombre des variables doivent correspondre à ceux attendus par la fonction.

Une variable passée en paramètre peut-être de type entier, réel ou chaine de caractère. Cela peut être aussi un tableau ou un élément d'un tableau à l'exclusion des tableaux virtuels ou des éléments de tableaux virtuels.

Il est important que le type de chaque variable corresponde à celui attendu par la fonction de la librairie. En effet aucune vérification n'est effectuée par SBASIC.

Ces sont les adresses des variables qui sont passées. Cela permet à la fonction d'accéder au contenu de la variable avant l'appel et d'en modifier éventuellement le contenu.

Si la valeur retournée par la fonction est non nulle, elle traitée par SBASIC comme un numéro d'erreur.

Voir aussi : OPEN LIBRARY.

CHAIN *nom-fich* [*num-ligne*] — Instruction

Charge en totalité un programme et l'exécute, éventuellement à partir d'une ligne donnée.

nom-fich	Nom du programme à exécuter.
num-ligne	Numéro de la ligne où débute l'exécution

Exemple 1

```
1000 CHAIN "ESSAI"
```

Le programme « ESSAI » est chargé puis exécuté.

Exemple 2

```
1000 P$="ESSAI"
1010 CHAIN P$ 100
```

Le programme « ESSAI » est chargé et exécuté à partir de la ligne 100.

Action

L'instruction CHAIN effectue le chargement complet du programme *nom-fichier* puis l'exécute, soit à partir du début, si aucun numéro de ligne n'est spécifié, soit à partir de la ligne *num-ligne*.

Lors de l'exécution d'une instruction CHAIN, le programme initial est effacé, toutes les variables sont remises à zéro et les fichiers ouverts sont fermés.

Si *nom-fich* ne comporte pas d'extension, l'extension « BAC » est utilisée ; un programme source peut également être chaîné, à condition de préciser son extension (« BAS »). Bien entendu, le temps de chargement est plus long dans ce cas.

Voir aussi : BLOAD, LOAD, RUN.

CHD *chemin-rep* — Instruction

Change de répertoire courant.

chemin-rep Chemin répertoire.

Exemple 1

```
10 CHD "Mes Programmes"
```

Cette ligne positionne le répertoire courant sur le sous-répertoire « Mes Programmes » situé dans le répertoire courant précédent.

Exemple 2

```
10 CHD "C:\SBASIC\Mes Programmes"
```

Cette ligne positionne le répertoire courant sur le sous-répertoire « \SBASIC\Mes Programmes » du disque C:.

Action

L'instruction CHD positionne le répertoire courant suivant le chemin spécifié. Ce chemin peut être absolu ou relatif.

CHR$(*exp-num*) — Fonction

Renvoie une chaîne d'un caractère dont le code ASCII est celui spécifié.

exp-num Code ASCII (compris entre 0 et 255).

Exemple

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

Cette instruction affiche la lettre « A » dans la fenêtre SBASIC.

Action

La fonction CHR$ retourne une chaîne d'un seul caractère dont le code ASCII a pour valeur celui de *exp-num*.

exp-num est tronquée à une valeur entière. Si cette valeur est négative ou supérieure à 255, une *erreur 74* est renvoyée.

Les codes ASCII de 0 à 31 ne sont pas affichables. Ce sont les caractères de contrôle. Les codes suivants sont gérés dans la fenêtre SBASIC :

Décimal	Hexadécimal	Action
7	07	Emission d'un bip
8	08	Déplacement du curseur à gauche
9	09	Déplacement du curseur vers la tabulation suivante
10	0A	Déplacement du curseur vers le bas

11	0B	Déplacement du curseur vers le haut
12	0C	Positionnement du curseur en haut à gauche
13	0D	Retour en début de ligne
27	1B	Caractère d'échappement 'Esc'
29	1D	Déplacement curseur vers la droite
Esc 0	1B 30	Sélection couleur de fond pour le texte
Esc 1	1B 31	Sélection couleur rouge pour le texte
Esc 2	1B 32	Sélection couleur verte pour le texte
Esc 3	1B 33	Sélection couleur jaune pour le texte
Esc 4	1B 34	Sélection couleur bleue pour le texte
Esc 5	1B 35	Sélection couleur magenta pour le texte
Esc 6	1B 36	Sélection couleur cyan pour le texte
Esc 7	1B 37	Sélection couleur blanche pour le texte
Esc 8	1B 38	Sélection couleur par défaut pour le texte
Esc H	1B 48	Inversion couleurs caractère et fond
Esc I	1B 49	Rétablissement couleurs caractère et fond
Esc p	1B 70	Effacement ligne sous le curseur
Esc q	1B 71	Effacement écran

Tableau 8 – Caractères de contrôle entre 0 et 31

Le code hexadécimal d'un caractère ASCII dans le tableau suivant s'obtient en juxtaposant le caractère de la colonne avant le 'x' et le caractère de la ligne après le 'x'. Exemple pour 'a' : 6x et x1 donnent le code hexadécimal 61 (97 en décimal).

	x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	xA	xB	xC	xD	xE	xF
2x		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3x	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4x	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5x	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6x	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7x	p	q	r	s	t	u	v	w	x	y	z	{	¦	}	~	⌂
8x	Ç	ü	é	â	ä	à	å	ç	ê	ë	è	ï	î	ì	Ä	Å
9x	É	æ	Æ	ô	ö	ò	û	ù	ÿ	Ö	Ü	ø	£	Ø	×	ƒ
Ax	á	í	ó	ú	ñ	Ñ	ª	º	¿	®	¬	½	¼	¡	«	»
Bx	▒	▒	▒	|	┤	Á	Â	À	©	╣	‖	╗	╝	¢	¥	┐
Cx	└	┴	┬	├	─	┼	ã	Ã	╚	╔	╩	╦	╠	=	╬	¤
Dx	ð	Ð	Ê	Ë	È	ı	Í	Î	Ï	┘	┌	▒	▄	¦	Ì	▀
Ex	Ó	ß	Ô	Ò	õ	Õ	µ	þ	Þ	Ú	Û	Ù	ý	Ý	¯	´
Fx	-	±	=	¾	¶	§	÷	¸	º	¨	·	¹	³	²	▪	

Tableau 9 – Caractères ASCII entre 32 et 255

Voir aussi : ASC.

CLEAR [*var-1*[(*)]] [,*var-2*[(*)]], … Instruction

Libère la place occupée par des variables ou des tableaux.

var-n[(*)] Variable ou tableau

Exemple 1

```
10 DIM B$(10), D(10,2): A$="CECI EST LE TITRE"
20 FOR X%=1 TO 10
30 INPUT B$(X%),D(X%,1),D(X%,2)
40 NEXT X%
50 PRINT A$
60 FOR X%=1 TO 10
70 PRINT B$(X%),D(X%,1),D(X%,2)
80 NEXT X%
90 CLEAR A$, B$(*), D(*)
100 DIM B$(20),D(50)
```

Les tableaux B$(*) et D(*) sont dimensionnés puis utilisés dans le programme avec les instructions INPUT et PRINT ; la variable A$ est également initialisée puis imprimée. Cette variable ainsi que les deux tableaux sont ensuite remis à zéro (ligne 90), ce qui permet d'affecter une nouvelle dimension aux tableaux (ligne 100).

Exemple 2

```
10 CLEAR
```

Toutes les variables et tous les tableaux du programme sont remis à zéro.

Action

L'instruction CLEAR remet à zéro tout ou partie des variables et tableaux du programme en cours.

Les arguments associés à l'instruction CLEAR peuvent être soit des variables soit des tableaux. Les tableaux, même s'ils possèdent plusieurs dimensions, sont référencés par leur nom auquel est accolé un seul astérisque placé entre parenthèses.

Les variables numériques prennent la valeur zéro et les chaînes deviennent des chaînes vides. La place occupée par les variables mentionnées est libérée, mais leur nom est conservé dans la table des symboles. Les tableaux ayant fait l'objet d'un ordre CLEAR peuvent être redimensionnés.

CLEAR sans argument remet à zéro l'ensemble des variables et tableaux du programme. Tous les fichiers sont fermés ainsi que la fenêtre graphique. Les effets des instructions WINDOW et SETCOLOR sur la fenêtre SBASIC sont annulés. Les variables XPEN et YPEN sont remises à leur valeur par défaut.

En mode direct, les arguments sont ignorés et la commande CONT devient ineffective.

Remarque

Il ne faut pas faire porter une instruction CLEAR sur un argument formel de sous-programme (SUB). Le résultat serait indéterminé.

Voir aussi : WINDOW, SETCOLOR, XPEN, YPEN, CONT.

CLOSE [*exp-num-1*, *exp-num-2*, …] Instruction

Ferme un ou plusieurs fichiers et libère les canaux internes correspondants.

exp-num-n Numéro de canal compris entre 1 et 32.

Exemple

```
10 OPEN OLD "ESSAI.DAT" AS 1
20 INPUT #1 A$
30 PRINT A$
40 CLOSE 1
```

La ligne 40 ferme le fichier ESSAI.DAT, ouvert sur le canal 1 (ligne 10).

Action

L'instruction CLOSE ferme les fichiers préalablement ouverts sur les canaux *exp-num*….

On peut fermer plusieurs fichiers dans un même ordre CLOSE, en spécifiant la liste des canaux, séparés par une virgule. Si l'un des canaux spécifiés n'est pas ouvert, l'instruction est sans effet, mais ne provoque aucune erreur. Lorsque l'instruction CLOSE est utilisée sans argument, tous les fichiers ouverts sont fermés.

La fermeture d'un fichier est nécessaire avant de pouvoir réutiliser le canal associé dans un ordre

OPEN.

Voir aussi : OPEN, OPEN NEW, OPEN OLD.

CLS — Instruction

Efface la fenêtre SBASIC.

Exemple

```
CLS
```

Action

Après l'exécution de cette commande, la fenêtre SBASIC s'efface et le curseur se positionne en haut à gauche de la fenêtre.

COMPILE *nom-fich* [,*exp-chaîne*,] [*num-ligne-1*, *num-ligne-2*, *num-ligne-3*] — Commande

Sauvegarde sous une forme précompilée tout ou partie d'un programme.

nom-fich	Nom de programme
exp-chaîne	Chaîne de caractères contenant les options C, S, P, O ou L
num-ligne-1	Première ligne à compiler.
num-ligne-2	Dernière ligne à compiler.
num-ligne-3	Nouveau numéro de la première ligne.

Exemple 1

```
COMPILE "ESSAI"
```

Le programme en mémoire est sauvegardé en totalité sous le nom « ESSAI.BAC », sous une forme précompilée, sans modification de sa numérotation.

Exemple 2

```
COMPILE "ESSAI" 2000,3000,2000
```

Le programme en mémoire est précompilé sous le nom « ESSAI.BAC », de la ligne 2000 à la ligne 3000, sans modification de sa numérotation (le nouveau numéro de la ligne 2000 est également 2000).

Exemple 3

```
COMPILE "ESSAI", "C", 2000,,100
```

Le programme en mémoire est précompilé sous le nom « ESSAI.BAC », de la ligne 2000 à la dernière ligne avec une renumérotation : la ligne 2000 porte désormais le numéro 100 et ainsi de suite. De plus, le programme est enregistré sous une forme compressée (option C).

Exemple 4

```
COMPILE "ESSAI", "CPS", ,1000,200
```

Le programme en mémoire est précompilé sous le nom « ESSAI.BAC », du début à la ligne 1000, avec une renumérotation : la ligne 1 porte le numéro 200 et ainsi de suite. L'enregistrement est effectué sous une forme compressée (option C), protégée (P) et la table des symboles est supprimée (S).

Action

La commande COMPILE sauvegarde sous une forme précompilée la portion du programme *nom-fich*, comprise entre *num-ligne-1* et *num-ligne-2*, en commençant la numérotation du programme sauvegardé à *num-ligne-3*.

Les valeurs par défaut de chacun des trois numéros de lignes optionnels sont les suivantes :

num-ligne-1	1
num-ligne-2	2147483647
num-ligne-3	*num-ligne-1*

Une ou plusieurs des options suivantes peuvent être utilisées :

- C (Compression) : les espaces et les remarques (REM) sont supprimées.
- S (Symboles) : suppression de la table des symboles. Le programme devient illisible.
- P (Protection) : le programme ne peut plus être listé ; il ne peut être qu'exécuté.
- O (Overlay) ou L(Librairie) : lors de son chargement, les lignes de ce programme s'ajoutent aux lignes déjà présentes. Seules les lignes dont le numéro se trouve entre la première ligne et la dernière ligne du programme compilé sont effacées.

Si l'option C, l'option S, l'option L ou un paramètre *num-ligne* est spécifiée, l'option O est forcée.

Si aucune option n'est spécifiée ou seulement l'option P, le programme se retrouve, une fois rechargé, dans son état d'origine. Les lignes déjà présentes sont effacées.

L'utilisation de la commande COMPILE plutôt que la commande SAVE permet d'obtenir un chargement du programme (avec l'instruction BLOAD) beaucoup plus rapide qu'avec l'instruction LOAD.

Voir aussi : SAVE, BLOAD.

CONT ou CONTINUE — Commande

Lance la poursuite d'un programme arrêté par une instruction STOP ou les touches Ctrl-C.

Exemple 1

```
10 PRINT "PREMIER": STOP: PRINT "DEUXIEME"
```

Lors de l'exécution de cette ligne, le mot « PREMIER » s'affiche, puis un message indique que le programme est arrêté :

```
STOP A LA LIGNE 10
```

Dès que l'on tape « CONT », l'exécution se poursuit et le mot « DEUXIEME » s'affiche à son tour.

Exemple 2

```
10 INPUT A$
20 PRINT A$
```

Si l'on tape Ctrl-C lors de l'exécution de l'instruction INPUT, un message signale l'arrêt du programme :

```
BREAK A LA LIGNE 10
```

La commande CONT permet de reprendre l'exécution à l'instruction INPUT.

Action

La commande CONTinue permet de reprendre l'exécution d'un programme interrompu soit par une instruction STOP, soit par appui sur Ctrl-C.

Si le programme est arrêté par une instruction STOP, la commande CONT fait reprendre l'exécution à l'instruction suivante ; s'il est arrêté par Ctrl-C alors qu'une instruction de type INPUT attend une entrée de données, CONT provoque la reprise de l'exécution à l'instruction INPUT elle-même.

La commande CONT ne peut pas faire redémarrer le programme dans les cas suivants :

- Si une erreur s'est produite au cours de l'exécution du programme ;
- Si la commande CLEAR est exécutée après l'interruption ;
- Si des lignes de programme référencées (ligne courante ou CALL, GOSUB ou FOR en cours) ont été modifiées ou supprimées après l'arrêt.

La commande CONT n'est utilisable qu'en mode direct.

Voir aussi : STOP.

COS(*exp-num*) — Fonction

Retourne le cosinus d'un angle.

exp-num	Angle exprimé en radians.

Exemple

```
10 PRINT COS (PI/3)
```

La valeur retournée est égale à 0,5, correspondant au cosinus de PI/3.

Action

La fonction COS renvoie la valeur du cosinus de l'angle *exp-num*.

exp-num représente un angle exprimé en radians.

Voir aussi : SIN.

CURSOR *exp-num-1*, *exp-num-2* — Instruction

Positionne le curseur en un point quelconque de la fenêtre SBASIC.

exp-num-1	Numéro de ligne (0 à 24).
exp-num-2	Numéro de colonne (1 à 80).

Exemple 1

```
10 CURSOR 10,40
20 PRINT "ESSAI"
```

Le message « ESSAI » s'affiche à partir de la 40e colonne de la 11e ligne.

Exemple 2

```
10 CURSOR
20 CURSOR 20,30
30 PRINT "POSITION DU CURSEUR : LIGNE "; YPEN, "COLONNE :"; XPEN,
40 CURSOR YPEN, XPEN
```

La position du curseur est donnée à la suite du message « POSITION DU CURSEUR… », qui s'affiche à partir de la 30e colonne de la 21e ligne. Puis le curseur est repositionné à l'emplacement qu'il occupait avant que le message n'apparaisse.

Action

Si les paramètres *exp-num-1* et *exp-num-2* sont spécifiés, l'instruction CURSOR permet de positionner le curseur en ligne *exp-num-1* et en colonne *exp-num-2*. Notez que les numéros de ligne et de colonne du caractère en haut à gauche de la fenêtre SBASIC sont respectivement 0 et 1. Ceci permet à SBASIC de supporter simplement les applications écrites pour un écran de 24 lignes de 80 caractères.

Sinon, l'instruction CURSOR a pour effet d'affecter aux variables système XPEN et YPEN respectivement, le numéro de ligne et le numéro de colonne sur lesquels est placé le curseur.

Le cas échéant, *exp-num-1* et *exp-num-2* sont tronquées à leur partie entière. Si la valeur entière de *exp-num-1* est supérieure à 24 ou si celle de *exp-num-2* est nulle ou supérieure à 80, une *erreur 97* se produit.

Si *exp-num-1* ou *exp-num-2* sont négatives, il se produit une *erreur 74.*

Si la fenêtre SBASIC est redimensionnée par l'instruction TEXT, les valeurs maximales de *exp-num-1* et *exp-num-2* évoluent en conséquence.

Voir aussi : XPEN, YPEN, TEXT.

CVTF$(*exp-réelle*) — Fonction

Transfère la valeur d'une expression numérique réelle dans une chaîne de caractères.

exp-réelle	Expression numérique réelle.

Exemple

```
10 A = 1200.56
20 A$ = CVTF$(A)
```

Après l'exécution de ce programme, la variable A$ se compose des huit octets correspondant à la représentation binaire de la variable réelle A.

Action

La fonction CVTF$ transfère le contenu de *exp-réelle* dans une variable chaîne de caractères.

Cette fonction ne réalise pas une conversion binaire - décimale, comme c'est le cas avec la fonction STR$, car la donnée numérique conserve sa représentation binaire dans la chaîne.

La fonction CVTF$ est surtout utilisée en liaison avec les instructions LSET, RSET et SET pour placer des nombres réels dans une mémoire tampon (définie au moyen des instructions FIELD ou FIELD#).

Voir aussi : CVT%$, CVT$%, CVT$F.

CVT%$(*exp-entière*) — Fonction

Transfère la valeur d'une expression numérique entière dans une chaîne de caractères.

exp-entière	Expression numérique entière.

Exemple

```
10 A% = 1200
20 A$ = CVT%$(A%)
```

Après l'exécution de ce programme, la variable A$ se compose des deux octets correspondant à la représentation binaire de la variable entière A.

Action

La fonction CVT%$ transfère le contenu de *exp-entière* dans une variable chaîne de caractères.

Cette fonction ne réalise pas une véritable conversion binaire - décimale, comme c'est le cas avec la fonction STR$, car la donnée numérique conserve sa représentation binaire dans la chaîne.

La fonction CVT%$ est surtout utilisée en liaison avec les instructions LSET, RSET et SET pour placer des entiers dans une mémoire tampon (définie au moyen des instructions FIELD ou FIELD#).

Voir aussi : CVTF$, CVT$%, CVT$F.

CVT$F(*var-chaîne*) Fonction

Renvoie la valeur d'un réel stocké dans une variable chaîne de caractères.

var-chaîne Variable chaîne de caractères.

Exemple

```
10 OPEN "PERSON" AS 1 LEN 50
20 FIELD #1, 30 AS NOM$, 12 AS PRE$, 8 AS SAL$
30 INPUT "Numéro (0=fin) ";N%
40 IF N% = 0 THEN 100
50 GET #1 RECORD N%
60 PRINT "Nom: "; NOM$
70 PRINT "Prénom: ";PRE$
80 PRINT "Salaire: ";CVT$F(SAL$) : PRINT
90 GOTO 30
100 CLOSE 1
```

Ce programme lit un enregistrement donné du fichier à accès direct « PERSON.DAT », contenant le nom, le prénom et le salaire d'un certain nombre de personnes. Dans le cas d'un fichier à accès direct, la communication entre le disque et la mémoire s'effectue obligatoirement par l'intermédiaire d'une mémoire tampon de type chaîne de caractères. Le salaire a donc été, lors de l'enregistrement, transféré au moyen de la fonction CVTF$. La fonction CVT$F (ligne 80) permet de le récupérer sous la forme d'un nombre réel.

Action

La fonction CVT$F renvoie le contenu de *var-chaîne* sous la forme d'un nombre réel.

Cette fonction est l'inverse de la fonction CVTF$. Elle permet de récupérer la valeur de l'expression numérique stockée dans une variable chaîne.

Voir aussi : CVTF$, CVT%$, CVT$%.

CVT$%(*var-chaîne*) Fonction

Renvoie la valeur d'un entier stocké dans une variable chaîne de caractères.

var-chaîne Variable chaîne de caractères.

Exemple

```
10 OPEN "PERSON" AS 1 LEN 44
20 FIELD #1, 30 AS NOM$, 12 AS PRE$, 2 AS AN$
30 INPUT "Numéro (0=fin) ";N%
```

```
40 IF N% = 0 THEN 100
50 GET #1 RECORD N%
60 PRINT "Nom: "; NOM$
70 PRINT "Prénom: ";PRE$
80 PRINT "Année de naissance: ";CVT$%(AN$) : PRINT
90 GOTO 30
100 CLOSE 1
```

Ce programme lit un enregistrement donné du fichier à accès direct « PERSON.DAT », contenant le nom, le prénom et l'année de naissance d'un certain nombre de personnes. Dans le cas d'un fichier à accès direct, la communication entre le disque et la mémoire s'effectue obligatoirement par l'intermédiaire d'une mémoire tampon de type chaîne de caractères. L'année de naissance a donc été, lors de l'enregistrement, transférée au moyen de la fonction CVT%$. La fonction CVT$% (ligne 80) permet de la récupérer sous la forme d'un nombre entier.

Action

La fonction CVT$% renvoie le contenu de *var-chaîne* sous la forme d'un nombre entier.

Cette fonction est l'inverse de la fonction CVT%$. Elle permet de récupérer la valeur de l'expression numérique entière stockée dans une *var-chaîne*.

Voir aussi : CVTF$, CVT%$, CVT$F.

DATA *const-1* [,*const-2*], … Instruction

L'instruction DATA introduit des données destinées à être lues ultérieurement par une instruction READ.

const-n	Constante numérique ou chaîne de caractères

Exemple

```
10 DATA JANVIER,31,FEVRIER,28,MARS,31
20 DATA AVRIL,30,MAI,31,JUIN,30
30 DATA JUILLET,31, AOUT,31, SEPTEMBRE,30
40 DATA OCTOBRE,31,NOVEMBRE,30,DECEMBRE,31
```

Les données placées après l'instruction DATA contiennent le nom de chaque mois et le nombre de jours qu'il comporte (les années non bissextiles).

Action

L'instruction DATA est destinée à placer à l'intérieur du programme des données de type alphabétique ou numérique pouvant être affectées ultérieurement à des variables de type correspondant au moyen de l'instruction READ.

Les données placées après une instruction DATA sont lues séquentiellement par des ordres READ ; l'affectation des données DATA aux variables des instructions READ est réalisée au moyen d'un *pointeur de DATA* :

- Au lancement du programme le *pointeur de DATA* pointe la première donnée de la première ligne DATA.
- Chaque fois qu'une instruction READ est exécutée, les éléments apparaissant dans la ligne DATA sont affectées aux variables associées à l'instruction READ, à partir de l'emplacement donné par le *pointeur de DATA* ; ce dernier avance alors du nombre de données lues.
- Si l'instruction READ n'exploite pas toutes les données de la ligne DATA, les données suivantes sont affectées aux futures instructions READ.

- Lorsque tout le contenu d'une ligne DATA a été lu, le *pointeur de DATA* passe à la ligne de DATA suivante et ainsi de suite.
- Le *pointeur de DATA* peut être réinitialisé au moyen de l'instruction RESTORE.

Si une chaîne de caractères contenant des espaces ou des virgules est nécessaire, elle doit être placée entre guillemets ou apostrophes.

L'instruction DATA doit être la première et la seule instruction dans une ligne de programme. Elle n'est pas utilisable en mode direct. Les lignes contenant l'instruction DATA peuvent être placées après l'instruction READ ; il n'est pas nécessaire que le déroulement du programme passe par ces lignes.

L'instruction DATA ne réalise aucune opération effective.

Voir aussi : READ, RESTORE.

DATE$ — Variable système

Fournit la date et l'heure courante.

Exemple

```
10 PRINT DATE$
```

Cette ligne affiche la date courante. Par exemple : 25/08/13 15:30:09.00

Action

DATE$ est une variable système (de même que ARGC, ARGV$, DATE$, ERR, ERL, PI, XPEN, YPEN) contenant la date courante sous forme d'une chaîne de caractères ainsi constituée :

JJ/MM/AA hh:mm:ss.cc

Où « JJ » représente le jour, « MM » le mois, « AA » l'année, « hh » l'heure, « mm » les minutes, « ss » les secondes et « cc » les centièmes.

DATE$ ne peut pas figurer à gauche du signe égal « = » dans un ordre d'affectation de données (LET explicite ou implicite).

DELETE [*num-ligne-1*] [,*num-ligne-2*] — Instruction

Supprime une zone du programme.

num-ligne-1	Première ligne à supprimer.
num-ligne-2	Dernière ligne à supprimer.

Exemple 1

```
10 DELETE 1000
```

Cette instruction supprime la ligne 1000 du programme courant.

Exemple 2

```
10 DELETE 1000,1100
```

Cette instruction efface les lignes 1000 à 1100 du programme courant.

Exemple 3

```
10 DELETE 2000,
```

Dans cet exemple, toutes les lignes depuis 2000 jusqu'à la dernière sont supprimées.

Exemple 4

```
DELETE ,100
```

Les premières lignes du programme sont effacées jusqu'à la ligne 100.

Action

L'instruction DELETE supprime les lignes comprises entre *num-ligne-1* et *num-ligne-2* du programme. Cette instruction s'utilise en mode direct ou en mode programme. Lorsqu'elle est utilisée en mode programme, une erreur 89 est signalée si une des lignes à effacer est référencée (ligne courante ou CALL, GOSUB ou FOR en cours).

La valeur par défaut de *num-ligne-1* est 1 ; celle de *num-ligne-2* est 2147483647.

DELETE est utilisée pour nettoyer la zone où l'on souhaite charger un nouveau sous-programme avec LOAD.

Voir aussi : NEW, LOAD.

DIGITS *exp-num-1* [,*exp-num-2*] [,*exp-num-3*] — Instruction

Précise le nombre de chiffres significatifs que l'on veut obtenir en édition.

exp-num-1	Nombre total de chiffres significatifs (entre 1 et 17).
exp-num-2	Nombre de chiffres après la virgule (optionnel).
exp-num-3	Nombre de zéros après la virgule, pour un nombre réel inférieur à 1 en valeur absolue, déclenchant l'affichage scientifique (optionnel).

Exemple 1

```
10 DIGITS 4,3
20 PRINT PI
```

PI est affiché sous la forme : 3.142

Exemple 2

```
10 DIGITS 1
20 PRINT 10
```

Lors de l'exécution, le nombre 10 apparaîtra sous la forme : 1E+1

Action

L'instruction DIGITS précise le nombre de chiffres significatifs que l'on veut obtenir en édition avec l'ordre PRINT ou lorsque l'on utilise l'instruction STR$. Elle n'a pas d'effet sur les résultats obtenus avec l'instruction PRINT USING.

exp-num-1 correspond au nombre total de chiffres significatifs, *exp-num-2* au nombre de chiffres après la virgule et *exp-num-3* au nombre de zéros après la virgule, pour un nombre réel inférieur à 1 en valeur absolue, déclenchant l'affichage scientifique. Le maximum de chiffres significatifs est 17 et le minimum est 1. Dans certains cas, l'utilisation du maximum 17 peut produire une 17e décimale erronée du fait de la conversion binaire - décimal. Il est donc déconseillé de demander le maximum de chiffres significatifs.

Le nombre de chiffres après la virgule doit toujours être inférieur au nombre total de chiffres significatifs, sinon une *erreur 73* se produit.

Le nombre de zéros après la virgule, pour un nombre réel inférieur à 1 en valeur absolue, déclenchant

l'affichage scientifique doit toujours être inférieur au nombre de chiffres après la virgule, sinon une *erreur 73* se produit.

L'instruction DIGITS n'est pas modifiée par une commande NEW ou RUN. Pour revenir aux valeurs par défaut au lancement de SBASIC, tapez ;

```
DIGITS 13,,2
```

Voir aussi : PRINT USING, STR$.

DIM *var* (*exp-num-1* [,*exp-num-2*] …) … Instruction

Permet de réserver l'espace mémoire des tableaux.

var	Nom du tableau.
exp-num-1	Première dimension du tableau.
exp-num-n	Enième dimension du tableau.

Exemple 1

```
10 DIM T$(10), A(100), A$(10,20), T%(4,4,3)
```

Cette ligne dimensionne 4 tableaux : T$(*) pourra comprendre jusqu'à 11 éléments ; A(*) jusqu'à 101 A$(*) jusqu'à 11 x 21 et T%, quant à lui, est un tableau d'entiers à 3 indices comprenant 100 éléments.

Exemple 2

```
10 INPUT "NOMBRE D'ELEMENTS"; N%
20 DIM B(N%)
30 FOR X% = 0 TO N%
40 PRINT "ELEMENT ";X%,
50 INPUT B(X%)
60 NEXT X%
```

Le tableau B(*) est dimensionné à la valeur de la variable N% saisie. Tous les éléments du tableau de B(0) à B(N%) sont ensuite saisis.

Action

L'instruction DIM réserve l'espace mémoire des tableaux mentionnés :

- Si plusieurs tableaux sont dimensionnés par une même instruction, leur nom doit être séparé par une virgule.
- Les tableaux doivent obligatoirement être dimensionnés avant que leurs éléments puissent être utilisés dans un programme. Un tableau ne peut être dimensionné qu'une seule fois dans un programme, sinon une *erreur 88* est renvoyée, sauf s'il a fait l'objet d'une instruction CLEAR. Dans ce cas, il doit de nouveau être dimensionné si l'on souhaite le réutiliser.
- On peut utiliser 0 comme indice d'un tableau.
- Le nombre d'indices d'un tableau n'est pas limité.
- L'espace mémoire réservé pour chaque élément d'un tableau est fonction du type de la variable :

Entier :	quatre octets.
Réel :	huit octets.
Chaîne de caractères :	huit octets, plus un nombre d'octets égal à la longueur de la chaîne.

Voir aussi : CLEAR.

DIM #*exp-num-1 var* (*exp-num-2* [,*exp-num-3*] …) [= *exp-num-4*] — Instruction

Permet de dimensionner un tableau virtuel et éventuellement de préciser la longueur de chaque élément d'un tableau virtuel de chaînes de caractères.

exp-num-1	Numéro de canal compris entre 1 et 32.
var	Nom du tableau.
exp-num-2	Première dimension du tableau.
exp-num-3	Deuxième dimension du tableau.
exp-num-4	Longueur de chaque élément d'un tableau de chaînes.

Exemple 1

```
10 OPEN "VIRT" AS 1
20 DIM #1 A(100)
```

Le fichier virtuel « VIRT » comprend jusqu'à 101 réels.

Exemple 2

```
10 OPEN "NOM" AS 1
20 DIM #1 N$(500) = 30
```

Le fichier virtuel « NOM » peut comprendre jusqu'à 501 chaînes ayant toutes une longueur de 30 caractères.

Action

L'instruction DIM# permet de dimensionner un tableau virtuel ouvert sur le canal *exp-num-1* :

- Un tel tableau doit obligatoirement être dimensionné avant que ses éléments puissent être utilisés dans le programme. Un tableau virtuel ne peut être dimensionné qu'une seule fois dans un programme, sinon une *erreur 88* est renvoyée, sauf s'il a fait l'objet d'une instruction CLOSE suivie d'un nouvel ordre OPEN.
- On peut utiliser 0 comme indice d'un tableau virtuel.
- Un tableau virtuel peut comprendre une ou deux dimensions.

Chaque élément d'un tableau virtuel d'entiers occupe sur disque un nombre d'octets qui dépend de son type :

Entier :	quatres octets.
Réel :	huit octets.
Chaîne de caractères :	nombre d'octets défini par *exp-num-4*, ou à défaut 18.

DPEEK *exp-num* — Fonction

Retourne le contenu de quatre octets de la mémoire.

exp-num	Adresse mémoire.

Exemple 1

```
10 A = DPEEK (HEX("24"))
```

Après l'exécution de cette ligne, la variable réelle A contient la valeur des octets d'adresse hexadécimale 24 à 27.

Action

La fonction DPEEK retourne le contenu des octets d'adresse *exp-num* à *exp-num*+3.

- Si *exp-num* est négative ou supérieure à 2147483647, une *erreur 75* se produit.
- La valeur retournée est la concaténation des octets d'adresse *exp-num*, d'adresse *exp-num*+1, d'adresse *exp-num*+2 et d'adresse *exp-num*+3, le premier étant le poids faible et le quatrième le poids fort.

Valeur = (((PEEK(*exp-num*+3) * 256 + PEEK(*exp-num*+2)) * 256 + PEEK(*exp-num*+1)) * 256 + PEEK(*exp-num*)

Voir aussi : PEEK, POKE, DPOKE.

DPOKE *exp-num-1*, *exp-num-2* — Fonction

Écrit quatre octets à une adresse mémoire.

exp-num-1	Adresse mémoire.
exp-num-2	Données.

Exemple

```
10 DPOKE 1000,3300
20 DPOKE L,X
30 DPOKE HEX('0123'),65000
```

Ce programme présente quelques utilisations possibles de l'instruction DPOKE.

Action

L'instruction DPOKE écrit la valeur de *exp-num-2* dans les 4 octets d'adresse *exp-num-1*, *exp-num-1*+1, *exp-num-1*+2 et *exp-num-1*+3. L'octet d'adresse *exp-num-1* reçoit l'octet de poids faible de cette valeur et l'octet d'adresse *exp-num-1*+3 reçoit l'octet de poids fort.

- *exp-num-1* doit être comprise entre 0 et 2147483647.
- *exp-num-2* doit être comprise entre -2147483648 et 2147483647.

Cette instruction doit être utilisée avec beaucoup de précautions car elle peut influer sur le fonctionnement de SBASIC.

Voir aussi : POKE, DPEEK, PEEK.

EDIT [*num-ligne*] — Commande

Permet de modifier une ligne du programme.

num-ligne	Numéro de ligne (entre 1 et 2147483647).

Exemple

```
EDIT 10
```

Cette commande affiche la ligne 10 avec le curseur placé en début de ligne.

Action

La commande EDIT permet de placer la ligne *num-ligne* en mode édition de manière à pouvoir la modifier. La valeur par défaut de *num-ligne* est celle de la dernière ligne créée, introduite ou listée. Le mode édition offre les fonctions suivantes :

Touche du clavier	Action
Ctrl-R	Déplace le curseur en début ou en fin de ligne (bascule).
Ctrl-C	Abandon de la modification.
Ctrl-X	Valide la ligne jusqu'au curseur.
↵ (Retour-Chariot)	Valide la ligne complète.
← (Flèche à gauche)	Recule le curseur d'un caractère.
→ (Flèche à droite)	Avance le curseur d'un caractère.
Echap	Efface la ligne.
Inser	Bascule le mode insertion.
Suppr	Efface le caractère placé sous le curseur (sans effet en fin de ligne).
← (Retour arrière)	Efface le caractère précédant le curseur.

Tableau 10 – Action des touches du clavier en mode d'édition d'un programme

Lorsqu'on entre une commande, une ligne de programme ou que l'on effectue une saisie par l'intermédiaire des instructions INPUT ou INPUT LINE, on utilise également l'éditeur.

La commande EDIT n'est utilisable qu'en mode direct.

END — Instruction

Termine l'exécution d'un programme.

Exemple

```
10 PRINT "DEBUT"
20 GOSUB ESSAI
30 END
40 LABEL ESSAI
50 PRINT "ESSAI"
60 RETURN
```

Le programme exécute les lignes 10 et 20 puis s'achève en ligne 30 lorsque l'instruction END est rencontrée.

Action

L'instruction END marque la fin d'un programme. Elle peut être placée n'importe où dans un programme.

Après une instruction END, les fichiers sont fermés et une commande CONT ne permet pas de redémarrer le programme.

Il n'est pas indispensable qu'une instruction END figure dans un programme.

Voir aussi : STOP.

ERL — Variable système

Variable système contenant le numéro de la ligne à laquelle s'est produite la dernière erreur.

Exemple

```
10 ON ERROR GOTO 60
20 PRINT " ENTREZ UNE CHAINE EXECUTABLE : "
30 INPUT LINE L$
40 EXECUTE L$
50 GOTO 20
60 IF ERL=40 THEN RESUME 20
70 PRINT "Erreur : ";ERR, "Ligne :" ;ERL
```

Ce programme demande la saisie d'une chaîne de caractères contenant des instructions exécutables. Si une *erreur 30* (types de données non concordants) se produit lors de l'exécution (ligne 30) de la chaîne saisie, le programme reprend en ligne 20.

Action

ERL est une variable système (de même que ARGC, ARGV$, ERR, DATE$, PI, XPEN et YPEN), contenant le numéro de la ligne à laquelle s'est produite la dernière erreur survenue.

ERL ne peut pas figurer à gauche du signe égal « = » dans un ordre d'affectation (LET implicite ou explicite).

Voir aussi : ERR, ON ERROR GOTO, RESUME.

ERR — Variable système

Variable système contenant le numéro de la dernière erreur rencontrée.

Exemple

```
10 ON ERROR GOTO 60
20 PRINT " ENTREZ UNE CHAINE EXECUTABLE : "
30 INPUT LINE L$
40 EXECUTE L$
50 GOTO 20
60 IF ERR=85 THEN PRINT "CHAINE VIDE": RESUME 20 ELSE IF ERR=86 THEN PRINT
   "CHAINE TROP LONGUE": RESUME 20
70 PRINT "Erreur : ";ERR, "Ligne: ";ERL
```

Ce programme demande la saisie d'une chaîne de caractères contenant des instructions exécutables. Si la chaîne est vide (*erreur 85*) ou trop longue (*erreur 86*), le message correspondant est affiché puis l'exécution reprend à la ligne 20.

Action

ERR est une variable système (de même que ARGC, ARGV$, ERL, DATE$, PI, XPEN et YPEN), contenant le numéro de la dernière erreur produite.

ERR ne peut pas figurer à gauche du signe égal « = » dans un ordre d'affectation (LET implicite ou explicite).

Voir aussi : ERL, ON ERROR GOTO, RESUME.

EXEC *exp-chaîne* — Instruction

Permet d'exécuter une ligne de commande Windows à partir de SBASIC.

exp-chaîne Chaîne de caractères contenant une ligne de commande.

Exemple

```
10 EXEC "DIR *.BAS"
```

L'exécution de cette ligne fait apparaître, dans la fenêtre SBASIC, la liste des fichiers ayant l'extension « .BAS » dans le répertoire courant.

Action

L'instruction EXEC permet d'exécuter une ligne de commande comme si elle avait été tapée directement au clavier dans une fenêtre de commande de Windows.

Voir aussi : « + », SYSTEM.

EXECUTE *exp-chaîne* — Instruction

Permet d'exécuter une ou plusieurs instructions SBASIC contenues dans une chaîne de caractères.

exp-chaîne Chaîne de caractères contenant un ou plusieurs ordres SBASIC.

Exemple 1

```
10 INPUT LINE A$
20 EXECUTE A$
```

Si, lors de l'exécution de ces lignes, l'utilisateur tape :

```
X=2 : X = 2 * X: PRINT X
```

La valeur 4 s'affichera dans la fenêtre SBASIC.

Exemple 2

```
10 A$ = ' 100 PRINT "NOUVELLE LIGNE " '
20 EXECUTE A$
```

Après exécution de ces lignes, la ligne 100 est créée dans le programme.

Action

L'instruction EXECUTE permet d'exécuter une suite d'ordres SBASIC contenus dans *exp-chaîne* comme s'ils étaient tapés directement au clavier. Cette instruction fait partie des particularités de SBASIC et offre un puissant moyen de génération automatique de programmes.

- Si *exp-chaîne* ne commence pas par un numéro de ligne, les instructions qu'elle contient sont exécutées immédiatement ;
- Si *exp-chaîne* commence par un numéro de ligne, celle-ci est ajoutée au programme et n'est exécutée que lorsque le déroulement du programme y passe. Si la ligne existe déjà, elle est remplacée. Une erreur 89 est signalée si la ligne à remplacer est référencée (ligne courante ou CALL, GOSUB ou FOR en cours).

EXIT — Commande

Permet de quitter SBASIC.

Exemple

```
EXIT
```

Dès la saisie de cette commande, on quitte SBASIC.

Action

La commande EXIT permet de quitter SBASIC. Elle n'est utilisable qu'en mode direct.

Voir aussi : SYSTEM.

EXP(*exp-num*) Fonction

Renvoie l'exponentielle d'une expression numérique.

exp-num	Expression numérique.

Exemple

```
10 PRINT EXP(1)
```

Ce programme affiche 2.718281828459, correspondant à la valeur de *e*, base du logarithme népérien.

Action

La fonction EXP retourne la valeur de l'exponentielle en base *e* de *exp-num*.

e vaut approximativement 2.718281828459045…

Si *exp-num* est supérieure à +355 ou inférieure à -355, une *erreur 102* se produit.

Voir aussi : LOG.

FIELD *exp-num-1* AS *var-chaîne-1* [,*exp-num-2* AS *var-chaîne-2*] … Instruction

Permet de définir la longueur et le nom d'une ou plusieurs zones de mémoire tampon.

exp-num-n	Longueur du tampon.
var-chaîne-n	Variable chaîne de caractères.

Exemple

```
10 FIELD 100 AS A$, 150 AS B$, 100 AS C$
```

Cette ligne de programme crée en mémoire un tampon d'une longueur de 350 caractères, répartis en trois zones. La première zone est référencée par la variable A$ et a pour longueur 100, la seconde est référencée par la variable B$ et a pour longueur 150, et la troisième est référencée par la variable C$ et a pour longueur 100 octets.

Action

L'instruction FIELD permet de définir la longueur et le nom de zones tampon.

exp-num-n indique la longueur de la zone affectée à *var-chaîne-n*.

L'affectation d'une valeur à une variable déclarée par une instruction FIELD ne peut être faite que par les instructions SET, LSET ou RSET. Toute affectation par une instruction LET (implicite ou explicite) détruit le lien entre la variable et le tampon.

Voir aussi : FIELD#, SET, LSET, RSET.

FIELD #*exp-num-1*, *exp-num-2* AS *var-chaîne-2* [,*exp-num-3* AS *var-chaîne-3*] … Instruction

Permet de définir la longueur et le nom des zones de mémoire tampon d'un fichier à accès direct.

exp-num-1	Numéro de canal compris entre 1 et 32.
exp-num-2	Longueur de la première zone du tampon.
var-chaîne-2	Nom de la première zone du tampon.
exp-num-3	Longueur de la deuxième zone du tampon.
var-chaîne-3	Nom de la deuxième zone du tampon.
…	

Exemple

```
10 OPEN "PERSON" AS 1, LEN 50
20 FIELD #1, 30 AS NOM$, 12 AS PRE$, 8 AS SAL$
30 INPUT "Numéro (0=fin) ";N%
40 IF N% = 0 THEN 100
50 GET #1, RECORD N%
60 PRINT "Nom: "; NOM$
70 PRINT "Prénom: "; PRE$
80 PRINT "Salaire: "; CVT$F (SAL$) : PRINT
90 GOTO 30
100 CLOSE 1
```

La ligne 20 découpe en trois zones le tampon d'entrée-sortie du fichier à accès direct « PERSON.DAT » précédemment ouvert sur le canal 1 : NOM$ a une longueur de 30 caractères ; PRE$ de 12 caractères et SAL$ de 8 caractères.

Action

L'instruction FIELD# permet de définir la longueur et le nom des zones constituant le tampon d'entrée-sortie d'un fichier à accès direct ouvert sur le canal *exp-num-1*.

Cette instruction doit être exécutée avant toute lecture (GET) ou écriture (PUT) dans le fichier correspondant.

exp-num-n indique la longueur du tampon affecté à *var-chaîne-n*.

L'affectation d'une valeur à une variable déclarée par une instruction FIELD# ne peut être faite que par les instructions SET, LSET ou RSET. Toute affectation par une instruction LET (implicite ou explicite) détruit le lien entre la variable et le tampon.

Voir aussi : FIELD, SET, LSET, RSET.

FOR *var-num* = *exp-num-1* TO *exp-num-2* [STEP *exp-num-3*] — Instruction

Marque le début d'une série d'instructions à exécuter en boucle.

var-num	Variable numérique compteur de boucle
exp-num-1	Valeur de départ du compteur de boucle.
exp-num-2	Borne supérieure du compteur de boucle.
exp-num-3	Pas d'incrémentation du compteur de boucle.

Exemple 1

```
10 FOR X = 1 TO 10
20 PRINT X
30 NEXT X
```

La ligne 10 marque le début d'une boucle : la variable X va prendre toutes les valeurs entières comprises entre 1 et 10 et la ligne 20 va être répétée autant de fois.

Exemple 2

```
10 FOR X = 10 TO 1 STEP -0.5
20 PRINT X
30 PRINT X/2
40 NEXT X
```

Ici, les instructions situées à l'intérieur de la boucle (lignes 20 et 30) seront exécutées pour les valeurs de X variant entre 10 et 1 avec un pas négatif de -0.5.

Action

Lorsqu'une instruction FOR est rencontrée, toutes les instructions comprises entre elle et l'instruction NEXT suivante sont exécutées pour toutes les valeurs de *var-num* comprises entre *exp-num-1* et *exp-num-2* avec un pas de *exp-num-3*.

La valeur par défaut de *exp-num-3* est 1.

- Une boucle FOR … NEXT se déroule de la façon suivante :
  1) Le compteur de la boucle *var-num* est initialisé à la valeur *exp-num-1*,
  2) Les instructions comprises entre l'instruction FOR et l'instruction NEXT suivante sont exécutées,
  3) La valeur du compteur de boucle *var-num*, augmentée de la valeur du pas *exp-num-3*, est comparée à la borne *exp-num-2* : si le pas est positif (respectivement négatif) et que le résultat est supérieur (respectivement inférieur) à la borne, alors l'exécution se poursuit à l'instruction qui suit l'instruction NEXT. Sinon, le pas est ajouté à compteur de boucle *var-num* et le processus reprend à l'étape 3.
- Le compteur de boucle ne doit normalement pas être modifié à l'intérieur de la boucle. Dans le cas contraire, c'est la nouvelle valeur qui serait utilisée à l'étape 3.
- La même variable doit être spécifiée comme compteur de boucle dans l'instruction FOR et dans l'instruction NEXT qui lui est associée. A défaut, une *erreur 62* est signalée.
- Il est interdit de se brancher directement à l'intérieur d'une boucle sans passer par l'instruction FOR. A défaut, une *erreur 62* serait détectée à l'exécution de l'instruction NEXT.

  Il est toutefois possible de sortir d'une boucle et de s'y rebrancher. Dans ce cas, les instructions exécutées à l'extérieur de la boucle peuvent être considérées comme faisant partie de la boucle. C'est, en particulier, ce qui a lieu lorsqu'un sous-programme est appelé de l'intérieur d'une boucle.

  Il est possible, et c'est un cas fréquent, de sortir définitivement d'une boucle avant avoir effectué toutes les itérations.
- Il est possible d'imbriquer plusieurs boucles à condition qu'elles ne se chevauchent pas : une boucle interne doit être entièrement contenue dans la boucle externe. De plus, le compteur de chaque boucle doit être distinct des autres compteurs

Voir aussi : NEXT.

FRE(*exp-num*) — Instruction

Retourne le nombre d'octets encore disponibles en mémoire.

exp-num	Identifiant mémoire.

Exemple

```
10 PRINT FRE(0)
```

Le nombre d'octets encore utilisables en mémoire données est affiché.

Action

La fonction FRE(*exp-num*) renvoie le nombre d'octets encore disponibles dans la mémoire spécifiée. Si *exp-num* est égale à 0 la mémoire données est identifiée. *exp-num* valant 1 identifie la mémoire symboles. Toute autre valeur de *exp-num* identifie la mémoire programme.

GET #*exp-num-1* [RECORD *exp-num-2*] — Instruction

Permet de lire un enregistrement dans un fichier à accès direct.

exp-num-1	Numéro de canal compris entre 1 et 32.
exp-num-2	Numéro d'enregistrement.

Exemple

```
10 OPEN "PERSON" AS 1, LEN 50
20 FIELD #1, 30 AS NOM$, 12 AS PRE$, 8 AS SAL$
30 INPUT "Numéro (0=fin) ";N%
40 IF N% = 0 THEN 100
50 GET #1, RECORD N%
60 PRINT "Nom: "; NOM$
70 PRINT "Prénom: "; PRE$
80 PRINT "Salaire: "; CVT$F (SAL$) : PRINT
90 GOTO 30
100 CLOSE 1
```

L'instruction GET de la ligne 50 permet de lire l'enregistrement N% du fichier à accès direct « PERSON.DAT », précédemment ouvert sur le canal 1.

Action

L'instruction GET permet de lire l'enregistrement *exp-num-2* du fichier à accès direct ouvert sur le canal *exp-num-1*.

- Si aucun fichier n'est ouvert sur le canal *exp-num-1*, une *erreur 43* se produit.
- Si *exp-num-2* ne correspond pas à un numéro d'enregistrement existant, une *erreur 24* est signalée.
- Si *exp-num-2* est omise, c'est l'enregistrement suivant l'enregistrement courant qui est lu.

Voir aussi : PUT.

GOSUB *adresse* — Instruction

Provoque un saut à un sous-programme.

adresse	Numéro de ligne ou étiquette.

Exemple 1

```
10 RAYON = 10: GOSUB 50
20 RAYON = 20: GOSUB 50
30 END
49 REM CALCUL DE LA CIRCONFERENCE D'UN CERCLE
50 CIRCONF = 2*PI*RAYON
60 PRINT "RAYON: "; RAYON,"CIRCONFERENCE: ";CIRCONF
70 RETURN
```

Les instructions GOSUB 50 des lignes 10 et 20 provoquent un saut au sous-programme débutant en ligne 50.

Exemple 2

```
10 RAYON = 10: GOSUB CERCLE
20 RAYON = 20: GOSUB CERCLE
30 END
49 REM CALCUL DE LA CIRCONFERENCE D'UN CERCLE
50 LABEL CERCLE: CIRCONF = 2*PI*RAYON
60 PRINT "RAYON: "; RAYON,"CIRCONFERENCE: ";CIRCONF
70 RETURN
```

Ici, le sous-programme appelé aux lignes 10 et 20 n'est pas référencé par un numéro de ligne, mais par une étiquette.

Action

L'instruction GOSUB provoque un branchement à un sous-programme dont l'adresse est soit un numéro de ligne soit une étiquette.

Voir aussi : CALL, SUB, LABEL, RETURN.

GOTO *adresse* — Instruction

Provoque un branchement inconditionnel à une ligne spécifiée.

adresse	Numéro de ligne ou étiquette.

Exemple 1

```
10 PRINT "DEBUT"
20 GOTO 50
30 REM LE PROGRAMME NE PASSE PAS ICI
40 REM LE PROGRAMME NE PASSE PAS ICI NON PLUS
50 PRINT "SUITE"
```

Lors de l'exécution, le programme « saute » directement de la ligne 20 à la ligne 50, ignorant les lignes 30 et 40.

Exemple 2

```
10 PRINT "ICI"
20 GOTO LABAS
30 REM LE PROGRAMME NE PASSE PAS ICI
40 REM LE PROGRAMME NE PASSE PAS ICI NON PLUS
50 LABEL LABAS
```

L'instruction GOTO LABAS de la ligne 20 provoque un branchement à la ligne référencée par l'étiquette LABEL.

Action

L'instruction GOTO effectue un branchement inconditionnel à l'adresse indiquée, qui peut être soit un numéro de ligne soit une étiquette.

L'instruction GOTO doit toujours être placée en fin de ligne sur une ligne comportant plusieurs instructions, parce qu'elle provoque immédiatement le branchement et que les instructions qui la suivent sur la même ligne ne sont pas exécutées.

Si l'adresse n'existe pas, une *erreur 60* se produit.

Voir aussi : IF GOTO, ON GOTO.

HEX(*exp-chaîne*) — Fonction

Convertit une chaîne de caractères hexadécimaux en la valeur correspondante.

exp-chaîne	Chaîne de caractères hexadécimaux en majuscules. (caractères 0 à 9 et A à F).

Exemple 1

```
10 PRINT HEX("100")
```

Ce programme affiche la valeur 256, correspondant à 100 en hexadécimal.

Exemple 1

```
10 A$="100"
20 PRINT HEX(A$)
```

Ce programme est équivalent au précédent, sauf qu'ici l'argument chaîne n'est pas une constante mais une variable.

Action

La fonction HEX retourne la valeur de *exp-chaîne* contenant un nombre hexadécimal en majuscules constitué des caractères 0 à 9 et A à F. L'évaluation s'arrête au premier symbole non hexadécimal rencontré.

Si *exp-chaîne* comporte plus de quatre caractères, HEX ne prend en compte que les quatre derniers.

IF *exp-logique* GOTO *adresse* — Instruction

Provoque un branchement conditionnel à une adresse spécifiée.

exp-logique	Expression logique à évaluer.
adresse	Numéro de ligne ou étiquette.

Exemple 1

```
100 IF Z%>B% GOTO 400
200 PRINT "Z% EST INFERIEUR OU EGAL A B%"
300 END
400 PRINT "Z% EST SUPERIEUR A B%"
```

Si la valeur de la variable Z% est supérieure à celle de la variable B%, le programme effectue un branchement à la ligne 400 ; sinon, l'exécution se poursuit à la ligne suivante (ligne 200).

Exemple 2

```
100 INPUT "VOULEZ-VOUS CONTINUER O/N "; A$
200 IF A$="O" GOTO SUITE
300 END
400 LABEL SUITE
500 PRINT"SUITE DU PROGRAMME"
```

Si la variable A$, saisie au clavier, contient la lettre « O » (en majuscule), l'exécution du programme se poursuit à l'adresse « SUITE », sinon le programme s'arrête à la ligne 300 (END).

Action

exp-logique est évaluée. Si elle est vraie, le programme se branche à l'adresse qui suit le mot-clé GOTO ; si elle est fausse, l'exécution se poursuit en séquence à la ligne suivante.

adresse est soit un numéro de ligne, soit une étiquette.

Voir aussi : IF GOSUB, IF THEN, IF THEN ELSE, ON GOTO.

IF *exp-logique* THEN *instr-comp* — Instruction

Exécute les instructions qui suivent la clause THEN si l'expression *exp-logique* est vraie.

exp-logique	Expression logique à évaluer.
instr-comp	Instruction(s) à exécuter.

Exemple

```
100 IF A=10 THEN 400
200 IF C+A<20 THEN C=C+10 : A=A+10
300 END
400 PRINT "A=10"
```

Si A est égal à 10, le programme effectue un branchement à la ligne 400 ; si A est différent de 10, l'exécution se poursuit en ligne 200 où un nouveau test est réalisé : si C+A est inférieur à 20, les variables A et C sont augmentées de 10.

Action

L'instruction IF permet d'orienter le déroulement d'un programme en fonction de la valeur, vraie ou fausse, d'une expression logique.

- *exp-logique* est évaluée. Si elle est vraie, *instr-comp* est exécutée ; si elle est fausse, l'exécution se poursuit en séquence à la ligne suivante.
- *instr-comp* est une instruction composée pouvant consister en plusieurs instructions séparées par le caractère « : ». Si elle comporte une instruction GOTO, celle-ci doit toujours être placée à la fin, car les autres instructions seraient ignorées.
  Si *instr-comp* se limite à « GOTO *num-ligne* », le mot-clé GOTO peut être omis.

Les instructions IF THEN peuvent être imbriquées entre elles ou avec des instructions IF THEN ELSE.

Voir aussi : IF GOTO, IF THEN ELSE.

IF *exp-logique* THEN *instr-comp-1* ELSE *instr-comp-2* — Instruction

---

Si l'expression logique est vraie, exécute les instructions suivant le mot-clé THEN ; si elle ne l'est pas, exécute les instructions suivant le mot-clé ELSE.

exp-logique	Expression logique à évaluer.
instr-comp-1	Instruction(s) à exécuter (expression vraie).
instr-comp-2	Instruction(s) à exécuter (expression fausse).

Exemple

```
100 IF H / (Y+5) < 8 THEN A = 5: GOTO LABAS ELSE PRINT "MAUVAIS"
200 END
300 LABEL LABAS
```

```
400 PRINT A
500 END
```

Lors de l'exécution de la ligne 100, l'expression logique « H/(Y+5)<8 » est évaluée : si elle est vraie, la valeur 5 est affectée à la variable A puis un branchement est effectué à l'étiquette LABAS ; si l'expression est fausse, le programme imprime « MAUVAIS » puis poursuit son exécution à la ligne suivante (END).

Action

L'instruction IF THEN ELSE permet d'orienter le déroulement d'un programme en fonction de la valeur, vraie ou fausse, d'une expression logique.

- *exp-logique* est évaluée. Si *exp-logique* est vraie, *instr-comp-1* est exécutée ; si *exp-logique* est fausse, *instr-comp-2* est exécutée.
- *instr-comp-1* et *instr-comp-2* sont des instructions composées pouvant consister en plusieurs instructions séparées par le caractère « : ». Si elles comportent une instruction GOTO, celle-ci doit toujours être placée à la fin, car les autres instructions seraient ignorées.
  Dans le cas où *instr-comp-1* ou *instr-comp-2* se limite à une instruction GOTO *num-ligne*, il est possible d'omettre le mot-clé GOTO.

Les instructions IF THEN ELSE peuvent être imbriqués entre elles ou avec des instructions IF THEN.

Voir aussi : IF THEN, IF GOTO.

INCH$(*exp-num-l* [,*exp-num-2*]) — Fonction

Permet l'entrée d'un ou plusieurs caractères du clavier ou d'un fichier séquentiel.

exp-num-1	Numéro de canal compris entre 0 et 32.
exp-num-2	Nombre de caractères à lire sur le canal.

Exemple 1

```
10 A$=INCH$(0)
20 PRINT A$
```

Ce programme attend qu'un caractère soit frappé au clavier (ligne 10) puis l'affiche (ligne 20).

Exemple 2

```
10 OPEN OLD "ESSAI.TXT" AS 1
20 FOR X%=1 TO 20
30 A$=INCH$(1)
40 PRINT A$ ;
50 NEXT X%
60 CLOSE 1
```

Ce programme lit (ligne 30) puis affiche (ligne 40) les 20 premiers caractères du fichier ESSAI.TXT, préalablement ouvert sur le canal 1.

Action

Cette fonction permet d'affecter à une variable chaîne, suivant la valeur de *exp-num-1*, soit un ou plusieurs caractères entrés au clavier soit des caractères lus dans un fichier séquentiel.

Après l'exécution, si *exp-num-2* n'est pas spécifié, la variable chaîne ne contient qu'un seul caractère ; lors d'une saisie au clavier, aucun Retour-Chariot n'est nécessaire pour valider ce caractère.

- Si *exp-num-2* est spécifié, après l'exécution la variable chaîne contient *exp-num-2* caractères.

exp-num-2 doit être supérieur ou égal à 1.

- Si *exp-num-1* est égale à 0, l'entrée se fait depuis le clavier ;
- Si *exp-num-1* est comprise entre 1 et 32, l'entrée se fait depuis le fichier séquentiel ouvert sur le canal *exp-num-1*. Si le numéro de canal indiqué ne correspond à celui d'aucun fichier ouvert, une *erreur 43* est signalée.
- Si *exp-num-1* est supérieure à 32, une *erreur 40* est signalée.

Voir aussi : INCH$(-1), INPUT, INPUT #, INPUT LINE, INPUT LINE #.

INCH$(-1) — Fonction

Permet de savoir si une touche du clavier a été tapée.

Exemple

```
10 A$ = INCH$(-1)
20 IF ASC(A$) = 0 THEN I = I + 1: GOTO 10
30 PRINT I
```

Ce programme ne s'arrête que lorsqu'une touche du clavier est tapée.

Action

Cette fonction teste le clavier sans interrompre l'exécution du programme : si une touche est tapée, elle renvoie le caractère correspondant ; sinon, elle retourne le caractère NUL, CHR$(0).

Voir aussi : INCH$.

INPUT [*const-chaîne*;]*var-1* [,*var-2*] … — Instruction

Permet d'afficher un message dans la fenêtre SBASIC puis de saisir des données au clavier et de les affecter à des variables.

const-chaîne	Constante chaîne de caractères.
var-n	Variable quelconque.

Exemple 1

```
10 INPUT "QUEL EST TON NOM ";N$
20 PRINT "BONJOUR ";N$
30 END
```

Lors de l'exécution de la ligne 10, le message « QUEL EST TON NOM » s'affiche, suivi d'un point d'interrogation engendré par l'instruction INPUT. Le programme attend que des données soient saisies puis validées par un Retour-Chariot. Les données sont alors affectées à la variable N$.

Exemple 2

```
10 PRINT " ENTREZ 5 NOMBRES "
20 INPUT A,B,C,D,E
```

L'ordre INPUT porte ici sur cinq variables. Chaque donnée introduite doit être séparée de la suivante par une virgule et la dernière doit être suivie d'un Retour-Chariot.

Action

L'instruction INPUT permet d'introduire au clavier des données qui sont affectées à des variables. Si une *chaîne* suit l'ordre INPUT, elle est d'abord affichée. L'instruction INPUT affiche ensuite un point

d'interrogation et attend que l'utilisateur saisisse la ou les données demandées.

Lorsque l'instruction INPUT attend plusieurs variables, chaque entrée au clavier doit être séparée de la suivante par une virgule. La dernière donnée doit être suivie d'un Retour-Chariot.

- Si le nombre de données introduites avant le Retour-Chariot est inférieur au nombre de données demandées, SBASIC affichera à nouveau un point d'interrogation, demandant ainsi les données manquantes.
- Si trop de données sont entrées, les données en trop sont ignorées.
- Contrairement à ce qui se produit avec l'instruction INPUT LINE, il n'est pas possible de n'entrer aucune donnée lors de l'exécution d'un ordre INPUT ; si tel était le cas, un nouveau point d'interrogation serait affiché.
- Les caractères point-virgule et virgule, entre les paramètres de l'instruction INPUT sont interchangeables.

L'utilisateur peut répondre à l'instruction INPUT par un appui sur Ctrl-C. Le programme est alors interrompu et SBASIC est prêt à recevoir d'autres commandes. Le programme reprendra son exécution à l'instruction INPUT si la commande CONT est frappée.

Voir aussi : INPUT LINE, INPUT #, CONT.

INPUT #0, [*const-chaîne*;] *var-1* [,*var-2*] … Instruction

Permet d'afficher un message non suivi d'un point d'interrogation puis d'entrer des données au clavier et de les affecter à des variables.

const-chaîne	Constante chaîne de caractères.
var-n	Variable quelconque.

Exemple 1

```
10 INPUT #0, "NOM: " ;N$
20 PRINT "BONJOUR " ;N$
30 END
```

Lors de l'exécution de la ligne 10, le message « NOM : » s'affiche, sans être suivi d'un point d'interrogation. Le programme attend qu'une chaîne soit introduite puis validée par un Retour-Chariot. Les données sont alors affectées à la variable N$.

Exemple 2

```
10 PRINT, " ENTREZ 5 NOMBRES "
20 INPUT #0, A,B,C,D,E
```

L'ordre INPUT porte cette fois-ci sur cinq variables. Là encore, aucun point d'interrogation n'est affiché lors de l'exécution de l'instruction INPUT.

Action

L'instruction INPUT #0 est identique à l'instruction INPUT à la différence près qu'elle n'affiche pas de point d'interrogation avant la saisie des données.

Voir aussi : INPUT, INPUT LINE, INPUT LINE #0.

INPUT #*exp-num* , *var-1* [,*var-2*] … Instruction

Permet de lire des données dans un fichier séquentiel et de les affecter à des variables.

exp-num	Numéro de canal compris entre 1 et 32.
var-n	Variable quelconque.

Exemple

```
10 OPEN OLD "PERSON" AS 1
20 INPUT #1 NOM$
30 INPUT #1 PRE$
40 INPUT #1 SAL
50 PRINT NOM$, PRE$, SAL
60 CLOSE 1
```

Les instructions INPUT #1 des lignes 20, 30 et 40 permettent de lire 3 données dans le fichier séquentiel « PERSON.DAT » précédemment ouvert sur le canal 1 (ligne 10) et de les affecter successivement aux variables chaînes NOM$ et PRE$ et à la variable réelle SAL.

Action

L'instruction INPUT # permet de lire des données dans un fichier séquentiel ouvert sur le canal *exp-num* et de les affecter à des variables.

Voir aussi : INPUT LINE #.

INPUT LINE *var-chaîne* — Instruction

Permet d'entrer au clavier une ligne complète dans une variable chaîne.

var-chaîne	Variable chaîne de caractères.

Exemple

```
10 INPUT LINE A$
```

Cette instruction permet d'entrer au clavier une ligne complète de texte contenant éventuellement des signes de ponctuation, des apostrophes ou des guillemets. La touche Retour-Chariot valide la saisie.

Action

L'instruction INPUT LINE est utilisée pour entrer au clavier une ligne complète dans une variable de type chaîne de caractères.

- La ligne entière est acceptée telle quelle, y compris les espaces et les signes de ponctuation éventuels, mais sans le Retour-Chariot. Aucun texte hormis le point d'interrogation « ? » n'est affiché préalablement comme c'est le cas avec l'instruction INPUT.
- Une chaîne vide peut être introduite.

Il est possible de taper Ctrl-C pour interrompre le programme ; la commande CONT permet alors de reprendre l'exécution à l'instruction INPUT LINE elle-même.

Voir aussi : INPUT, INPUT LINE #, CONT.

INPUT LINE #*exp-num, var-chaîne* — Instruction

Permet de lire dans une variable chaîne une ligne complète à partir d'un fichier séquentiel.

exp-num	Numéro de canal compris entre 0 et 32.
var-chaîne	Variable de chaîne de caractères.

Exemple

```
10 DIM LIGNE$(100)
20 ON ERROR GOTO 1000
30 OPEN OLD "FICHIER.TXT" AS 1
35 I%=0
40 INPUT LINE #1,LIGNE$(I%)
50 I%=I%+1
60 IF I%<=100 GOTO 40
70 CLOSE 1
80 FOR J%=0 TO I%-1
90 PRINT LIGNE$(J%)
100 NEXT J%
110 END
1000 IF ERR = 8 THEN PRINT I% ;"lignes lues": RESUME 70
1010 PRINT "Erreur";ERR;"à la ligne";ERL
```

Ce programme range les lignes du fichier « FICHIER.TXT » dans le tableau LIGNE$(*) puis affiche le nombre de lignes et le contenu du tableau.

Action

L'instruction INPUT LINE # est utilisée pour lire, à partir d'un fichier séquentiel, une ligne complète dans une variable de type chaîne de caractères.

- Une ligne entière est acceptée telle quelle, y compris les espaces et les signes de ponctuation éventuels, mais pas le Retour-Chariot.
- Si *exp-num* est 0, la lecture est effectuée depuis le clavier. L'instruction INPUT LINE #0 est en tous points identique à l'instruction INPUT LINE à la différence près qu'elle n'affiche pas un point d'interrogation avant la saisie des données.

Voir aussi : INPUT, INPUT #, INPUT LINE.

INSTR(*exp-num*, *exp-chaîne-1*, *exp-chaîne-2*) — Fonction

Recherche une sous-chaîne dans une chaîne, et renvoie sa position.

exp-num	Rang du caractère où débute la recherche.
exp-chaîne-1	Chaîne de caractères dans laquelle s'effectue la recherche.
exp-chaîne-2	Sous-chaîne recherchée.

Exemple 1

```
10 C$ = "0123456789"
20 S$ = "34"
30 A% = INSTR (1,C$,S$)
40 PRINT A%
50 B% = INSTR (5,C$,S$)
60 PRINT B%
```

Après l'exécution de ce programme, la variable A% contient la position, dans C$, à laquelle commence la sous-chaîne S$, à savoir 4. La variable B% sera, elle, égale à 0, car la sous-chaîne S$ n'existe pas dans la portion de C$ commençant au 5e caractère.

Exemple 2

```
10 I% = 1 : N% = 0 : C$ = "IL ETAIT UNE FOIS DANS L'OUEST"
```

```
20 A% = INSTR (I%,C$,"E")
30 IF A%>0 THEN N% = N% + 1 : I% = A% + 1 : GOTO 20
40 PRINT " LA LETTRE 'E' APPARAIT " ; N% ; " FOIS"
```

Ce programme totalise les occurrences de la lettre « E » dans la chaîne « IL ETAIT UNE FOIS DANS L'OUEST ».

Action

La fonction INSTR recherche la première occurrence de la sous-chaîne *exp-chaîne-2* dans la chaîne de caractères *exp-chaîne-1* à partir de la position *exp-num.*

- Si la valeur de *exp-num* est :
  - Inférieure ou égale à 0, une *erreur 52* est signalée ;
  - Omise, la recherche commence à la position 1 ;
  - Supérieure ou égale à 1, la recherche commence à la position spécifiée ;
  - Supérieure à la longueur de la chaîne dans laquelle s'effectue la recherche, INSTR renvoie la valeur 0 et aucune erreur n'est signalée.
- Si *exp-chaîne-2* est trouvée dans *exp-chaîne-1*, INSTR renvoie la position du premier caractère de *exp-chaîne-2* dans *exp-chaîne-1*
- Si *exp-chaîne-2* n'est pas trouvée, la valeur 0 est renvoyée.

INT(*exp-num*)	Fonction

Renvoie la partie entière d'une expression numérique.

exp-num	Expression numérique quelconque.

Exemple

```
10 FOR X = -1.5 TO 1.5 STEP 0.5
20 PRINT X, INT(X)
30 NEXT X
```

Ce programme affiche la valeur prise par la variable X ainsi que sa partie entière :

1.5	2
1	1
0	0
1	1
1.5	1

Action

La fonction INT retourne la partie entière de *exp-num*, c'est-à-dire le plus grand entier qui lui est inférieur ou égal.

KILL *nom-fich*	Instruction

Supprime un fichier disque.

nom-fich	Nom de fichier.

Exemple 1

```
10 KILL "ESSAI"
```

Cette ligne détruit le fichier « ESSAI.BAS » dans le répertoire courant.

Exemple 2

```
10 KILL "LETTRE.TXT"
```

Le fichier « LETTRE.TXT » est supprimé du répertoire courant.

Action

L'instruction KILL détruit le fichier désigné. L'extension « BAS » est affectée par défaut à *nom-fich* et le répertoire par défaut est le répertoire courant.

Voir aussi : RENAME, OPEN NEW.

LABEL *étiquette* — Déclaration

Permet de désigner une ligne au moyen une étiquette.

étiquette	Nom d'étiquette.

Exemple

```
10 INPUT X
20 IF X=1 GOTO SUITE
30 GOTO 10
40 LABEL SUITE
50 PRINT "suite "
60 END
```

Lors de l'exécution, si la variable X prend la valeur 1, le programme effectue un branchement à la ligne étiquetée « SUITE ».

Action

L'instruction LABEL permet de baptiser une ligne au moyen d'une *étiquette* de façon à pouvoir y accéder ultérieurement (par une instruction GOTO ou GOSUB) en indiquant ce nom plutôt que le numéro de la ligne.

- Le nom de l'étiquette se construit de la même manière que le nom des variables.
- L'instruction LABEL doit obligatoirement être la première d'une ligne à instructions multiples.

Voir aussi : GOTO, GOSUB, ON GOTO, ON GOSUB, SUB.

LEFT$(*exp-chaîne*, *exp-num*) — Fonction

Renvoie la partie gauche d'une chaîne de caractères.

exp-chaîne	Chaîne de caractères.
exp-num	Nombre de caractères.

Exemple

```
10 A$="0123456"
20 PRINT LEFT$(A$,3)
```

Ce programme imprime les 3 premiers caractères de la chaîne A$, à savoir, « 012 ».

Action

La fonction LEFT$ délivre une chaîne de caractères constituée des *exp-num* caractères de gauche de la

chaîne de caractères *exp-chaîne*.

- Si *exp-num* est supérieure à 0 et inférieure à la longueur de la chaîne de caractères *exp-chaîne*, la partie gauche de cette chaîne, composée du nombre de caractères spécifiés, est renvoyée,
- Si *exp-num* est supérieure ou égale à la longueur de la chaîne *exp-chaîne*, celle-ci est retournée en totalité,
- Si *exp-num* est nulle, une chaîne vide est renvoyée,
- Si *exp-num* est négative, une *erreur 52* est signalée,
- Si *exp-num* est supérieure à 2147483647 une *erreur 104* est signalée.

Voir aussi : RIGHT$, MID$.

LEN(*exp-chaîne*)	Fonction

Renvoie la longueur d'une chaîne de caractères.

exp-chaîne	Chaîne de caractères.

Exemple

```
10 A$="ABCDEF"
20 PRINT LEN(A$)
```

Ce programme affiche la longueur de la chaîne de caractères A$, à savoir 6.

Action

La fonction LEN renvoie le nombre de caractères contenus dans la chaîne de caractères *exp-chaîne*.

- Tous les caractères sont comptés, y compris les espaces et les caractères non affichables.
- Si *exp-chaîne* est vide, la valeur 0 est retournée.

LET *var* = *exp*	Instruction

Affecte une valeur à une variable.

var	Variable numérique (ou chaîne) à affecter.
exp	Expression numérique (ou chaîne).

Exemple

```
10 LET JOUR$ = "LUNDI"
20 LET A% = B% * 2 + 20
30 BB (1) = A
40 Z% = Z% + 1
```

Ce programme donne quelques exemples d'affectations possibles :

- Affectation d'une constante chaîne à une variable chaîne (ligne 10)
- Affectation d'une expression numérique à une variable entière (ligne 20)
- Affectation d'une variable réelle à un élément d'un tableau préalablement déclaré (ligne 30)
- Incrémentation d'une variable entière (ligne 40).

Action

L'instruction LET affecte une valeur à une variable.

- Toute variable peut se faire affecter une valeur par l'utilisation de cette instruction à l'exception

des variables associées à une zone tampon et déclarées au moyen d'une instruction FIELD ou FIELD#, sous peine de voir détruit le lien entre ces variables et le tampon.

L'expression à droite du signe égal est évaluée puis affectée à la variable dont le nom est indiqué à gauche du signe égal.

La valeur peut être une constante, une variable ou une expression complexe.

- L'instruction LET peut être omise.
- Dans le cas où une expression réelle est affectée à une variable entière, elle est convertie, sauf si elle est trop grande, auquel cas une *erreur 104* se produit.

LIST {[*num-ligne-1*] [,*num-ligne-2*] | *num-ligne-1*,} Commande

---

Liste tout ou partie du programme présent en mémoire.

num-ligne-1	Première ligne à lister.
num-ligne-2	Dernière ligne à lister.

Exemple 1

```
LIST
```

Liste tout le programme.

Exemple 2

```
LIST 10
```

Liste la ligne 10.

Exemple 3

```
LIST 50,80
```

Liste les lignes 50 à 80.

Exemple 4

```
LIST ,60
```

Liste de la première ligne à la ligne 60.

Exemple 5

```
LIST 100,
```

Liste de la ligne 100 à la dernière ligne.

Action

La commande LIST permet de lister tout ou partie du programme présent en mémoire.

- Le programme est listé de la ligne *num-ligne-1* à la ligne *num-ligne-2*.
- La valeur par défaut de *num-ligne-1* est 1 et celle de *num-ligne-2* est 2147483647.

Afin d'améliorer la visibilité du programme, les mots clé sont affichés en bleu, les variables en vert, les chaines de caractères en jaune et les commentaires en rouge.

LOAD *nom-fich* [*num-ligne-1*] [, *num-ligne-2*] [, *num-ligne-3*] Instruction

---

Charge en mémoire tout ou partie d'un programme sauvegardé sous forme texte, avec une translation optionnelle des numéros de lignes.

nom-fich	Nom du programme à charger.

num-ligne-1	Numéro de la première ligne à charger.
num-ligne-2	Numéro de la dernière ligne à charger.
num-ligne-3	Translation.

Exemple 1

```
10 LOAD "ESSAI"
```

Le programme « ESSAI.BAS » est chargé en totalité, sans modification de sa numérotation.

Exemple 2

```
10 LOAD "ESSAI "1000,
```

Le programme « ESSAI.BAS » est chargé de la ligne 1000 à la dernière ligne, sans modification de sa numérotation.

Exemple 3

```
10 LOAD "ESSAI" ,1000
```

Le programme « ESSAI.BAS » est chargé du début jusqu'à la ligne 1000, sans modification de sa numérotation.

Exemple 4

```
10 LOAD "ESSAI" 100,1000,200
```

Le programme « ESSAI.BAS » est chargé de la ligne 100 à la ligne 1000, avec une translation de 200 dans la numérotation : la ligne 100 du programme porte le numéro 300, et ainsi de suite.

Exemple 5

```
10 LOAD "ESSAI" , , 200
```

Le programme « ESSAI.BAS » est chargé en totalité, et 200 est ajouté à tous les numéros de ligne.

Action

L'instruction LOAD charge la portion du programme texte *nom-fich*, comprise entre *num-ligne-1* et *num-ligne-2*, en ajoutant *num-ligne-3* à chaque numéro de ligne chargée. L'extension par défaut de *nom-fich* est « BAS ». Le répertoire par défaut est le répertoire courant.

- Les valeurs par défaut de chacun des 3 paramètres optionnels sont les suivantes :

num-ligne-1	1
num-ligne-2	2147483647
num-ligne-3	0

- Lorsqu'un paramètre présent est précédé de paramètres omis, l'emplacement de ces derniers doit être marqué par des virgules (Exemples 3 et 5).
- En cas de conflit avec des lignes se trouvant en mémoire, ce sont les lignes du programme *nom-fich* qui s'imposent et prennent la place de leurs homologues déjà présentes en mémoire.
- L'instruction LOAD s'utilise en mode programme ou en mode direct. En mode programme, une erreur 89 est signalée si une ligne à remplacer est référencée (ligne courante ou CALL, GOSUB ou FOR en cours).

Voir aussi : BLOAD.

LOCAL *var-1* [,*var-2*, …] — Déclaration

Permet, dans un sous-programme, de rendre certaines variables distinctes des variables de même nom

utilisées dans le programme appelant.

var-n Variable numérique ou chaîne de caractères.

Exemple

```
10 A% = 10 : CALL ESSAI
20 PRINT A%
30 END
40 SUB ESSAI: LOCAL A%
50 FOR A% = 1 TO 3
60 PRINT A%
70 NEXT A%
80 RETURN
```

La valeur 10 est affectée à la variable A%. Cette variable est utilisée comme variable d'une boucle dans le sous-programme ESSAI et prend des valeurs comprises entre 1 et 3. Cependant, du fait de l'instruction LOCAL placée au début du sous-programme, la variable A% retrouve sa valeur initiale (10) après le retour (RETURN) au programme appelant.

Action

La déclaration LOCAL s'utilise **uniquement dans un sous-programme** pour rendre une ou plusieurs variables, distinctes des variables pouvant porter le même nom dans le programme appelant.

- LOCAL ne doit être exécutée qu'une fois à chaque passage dans le sous-programme, et ce, avant toute référence à l'une quelconque des variables qu'elle déclare. Une utilisation ne respectant pas ces règles est susceptible de donner des résultats imprévisibles.
- Une fois effectué le retour au programme appelant, le contenu des variables de même nom du programme appelant redevient accessible alors que celui des variables déclarées locales ne l'est plus.

Voir aussi : CALL, SUB.

LOG(*exp-num*) — Fonction

Retourne le logarithme népérien d'une expression numérique.

exp-num Expression numérique positive.

Exemple

```
10 PRINT LOG(EXP(1))
```

La valeur retournée est égale à 1, correspondant au logarithme népérien de *e*.

Action

La fonction LOG renvoie la valeur du logarithme népérien de *exp-num*.

Si *exp-num* est négative ou nulle, une *erreur 105* est signalée.

Voir aussi : EXP.

LPRINT [*exp-1*] [{ , | ; } *exp-2*] … [{ , | ; }] — Instruction

Écrit des données simultanément dans la fenêtre SBASIC et dans le tampon d'impression.

exp-n Expression chaîne ou numérique.

{ , | ; } Séparateur « , » ou « ; ».

Exemple

L'instruction LPRINT permet d'écrire simultanément dans la fenêtre SBASIC et dans le tampon d'impression. Elle est par ailleurs identique à l'instruction PRINT.

Voir aussi : LPRINT USING, PRINT, PRINT USING, TAB, DIGITS.

LSET *var-chaîne* = *exp-chaîne* — Instruction

Affecte une valeur à la partie gauche d'une zone de mémoire tampon, au besoin en la tronquant ou en la complétant par des espaces.

var-chaîne Variable chaîne de caractères.
exp-chaîne Expression chaîne de caractères.

Exemple

```
10 FIELD 8 AS A$, 10 AS B$
20 LSET A$="12345"
30 LSET B$="CECI EST UN TEST"
```

Après exécution de ces lignes, la variable A$ contient « 12345□□□ » et la variable B$ contient « CECI EST U ». En effet, la chaîne affectée à A$ étant plus courte que la longueur du tampon, elle a été complétée par des espaces ; au contraire, la chaîne affectée à B$ étant supérieure à la taille du tampon, elle a été tronquée.

Action

L'instruction LSET permet d'affecter une valeur à la partie gauche d'une zone de mémoire tampon définie par une instruction FIELD.

- *exp-chaîne* est placée dans le tampon associé à *var-chaîne* ; elle est justifiée à gauche.
- Si *exp-chaîne* est plus courte que la longueur du tampon, elle est complétée avec des espaces ; si elle est plus longue, elle est tronquée.

De même que les instructions SET et RSET, LSET est destinée à être utilisée avec des variables associées aux tampons d'entrée-sortie.

Voir aussi : FIELD, FIELD#, RSET, SET.

LTRIM$(*exp-chaîne*) — Fonction

Retourne une chaîne de caractères épurée des espaces de tête.

exp-chaîne Expression chaîne de caractères.

Exemple

```
10 A$ = "□□□CECI EST UN ESSAI□□□"
20 L$ = LTRIM$(A$)
```

Après l'exécution, L$ contient « CECI EST UN ESSAI□□□ »

Action

La fonction LTRIM$ renvoie le contenu de *exp-chaîne* débarrassé de ses espaces de tête.

Voir aussi : RTRIM$.

MID$(*exp-chaîne*, *exp-num-1*[, *exp-num-2*]) Fonction

Renvoie une portion d'une chaîne de caractères, commençant à une position donnée.

exp-chaîne	Chaîne de caractères.
exp-num-1	Position dans la chaîne.
exp-num-2	Nombre de caractères demandés (optionnel).

Exemple 1

```
10 A$="ABCDEF"
20 PRINT MID$(A$,3)
```

Ce programme affiche la partie de la variable A$ comprise entre le troisième caractère et la fin, à savoir « CDEF ».

Exemple 2

```
10 A$="ABCDEF"
20 PRINT MID$(A$,3,2)
```

Dans cet exemple, la partie de A$ qui est affichée par le programme commence également en troisième position, mais elle ne comprend que deux caractères : « CD ».

Action

La fonction MID$ renvoie la partie de *exp-chaîne* commençant à la position *exp-num-1* et comprenant *exp-num-2* caractères.

- Si *exp-num-1* ou *exp-num-2* est négative, une *erreur 52* est signalée,
- Si *exp-num-1* est nulle ou supérieure à la longueur de *exp-chaîne*, une chaîne vide est renvoyée,
- Si *exp-num-2* est omise, la partie droite de *exp-chaîne* débutant à la position *exp-num-1* est renvoyée,
- Si *exp-num-2* est nulle, une chaîne vide est renvoyée,
- Si *exp-num-2* est supérieure à la taille de la portion située à droite de la position *exp-num-1*, toute cette portion est renvoyée.

Voir aussi : LEFT$, RIGHT$, LEN.

NEW Commande

Efface le programme présent en mémoire.

Action

La commande NEW efface le programme présent en mémoire. La mémoire est alors prête à recevoir un nouveau programme.

Cette commande n'est utilisable qu'en mode direct.

Voir aussi : OPEN.

NEXT *var-num* Instruction

Indique la fin d'une boucle.

var-num Variable numérique.

Exemple 1

```
10 FOR X= 1 TO 10
20 PRINT " essai ";
30 PRINT X
40 NEXT X : PRINT "FIN"
```

Dans cet exemple, les lignes 20 et 30 sont exécutées pour toutes les valeurs de X comprises entre 1 et 10. Chaque fois que l'instruction NEXT de la ligne 40 est rencontrée, la valeur de la variable X est incrémentée de 1. Tant que cette valeur ne dépasse pas 10, l'exécution reprend à l'instruction 20. Lorsque X dépasse cette valeur, le programme passe à l'instruction suivant NEXT.

Action

L'instruction NEXT marque la fin d'une boucle initiée par une instruction FOR.

La variable *var-num* doit correspondre au nom de la variable de contrôle de l'instruction FOR qui précède, à défaut une *erreur 62* est signalée.

Voir aussi : FOR.

ON ERROR GOTO [*adresse*] — Instruction

Active une procédure de gestion d'erreur.

adresse Numéro de ligne ou étiquette.

Exemple

```
10 INPUT "TAPEZ LE NOM DU FICHIER AVEC EXTENSION"; F$
20 ON ERROR GOTO 100
30 OPEN OLD F$ AS 1
40 INPUT LINE#1, L$
50 PRINT L$
60 GOTO 40
70 PRINT "FIN DE FICHIER"
80 CLOSE 1
90 END
99 REM *** gestion des erreurs ***
100 IF ERR = 8 THEN RESUME 70
110 IF ERR = 4 THEN PRINT "FICHIER ABSENT": END
120 PRINT "ERREUR: "; ERR, "LIGNE: ";ERL: RESUME 10
```

Si une erreur survient lors de l'exécution de ce programme, un branchement est effectué à la ligne 100. Les lignes 100 à 120 constituent une procédure de gestion d'erreurs permettant, en fonction du type de l'erreur rencontrée (fichier absent, fin de fichier, …), d'effectuer le traitement approprié.

Action

L'instruction ON ERROR permet d'activer une procédure de gestion d'erreurs : si une erreur survient lors de l'exécution du programme, un branchement est immédiatement effectué à la ligne *adresse* identifiée par son numéro ou son étiquette.

Si *adresse* est omis ou nul, le programme s'arrête lorsqu'une erreur survient et affiche le numéro de l'erreur, le numéro de la ligne et le message correspondant.

Voir aussi : ERL, ERR, RESUME.

ON *exp-num* GOSUB *adresse-1* [,*adresse-2*] … Instruction

Appelle un sous-programme en fonction de la valeur d'une expression.

exp-num	Expression numérique entière.
adresse-n	Numéro de ligne ou étiquette.

Exemple

```
10 INPUT "CHOIX (1-5)"; A%
20 IF A%=0 THEN END ELSE IFA%>5 THEN 10
30 ON A% GOSUB 100,110,120,130,140
40 GOTO 10
100 PRINT "1": RETURN
110 PRINT "2": RETURN
120 PRINT "3": RETURN
130 PRINT "4": RETURN
140 PRINT "5": RETURN
```

Suivant la valeur de la variable A% saisie au clavier, le programme effectue un saut à l'un des sous-programmes des lignes 100 à 140.

Action

L'instruction ON GOSUB permet d'effectuer un saut conditionnel à un sous-programme en fonction de la valeur de *exp-num*.

- Si la valeur de *exp-num* est n, le saut s'effectue à la enième adresse indiquée dans la liste ; l'adresse du sous-programme appelé peut être définie par un numéro de ligne ou par une étiquette.
- Le cas échéant, *exp-num* est tronquée à sa partie entière. Si la valeur de celle-ci est nulle ou supérieure au nombre d'adresses spécifiées, une *erreur 32* est signalée ; si l'adresse sélectionnée n'existe pas, une *erreur 60* est produite.

Aucune instruction ne doit suivre ON GOTO dans les lignes à instructions multiples.

Voir aussi : GOSUB, ON GOTO.

ON *exp-num* GOTO *adresse-1* [,*adresse-2*] … Instruction

Effectue un branchement conditionnel à une ligne du programme.

exp-num	Expression numérique.
adresse-n	Numéro de ligne ou étiquette.

Exemple

```
10 INPUT "CHOIX (1-5)"; A%
20 IF A%=0 THEN END ELSE IFA%>5 THEN 10
30 ON A% GOTO 100,DEUX,200,QUATRE,130,140
40 GOTO 10
100 PRINT "1" : END
130 PRINT "5" : END
140 PRINT "6" : END
```

```
200 PRINT "3" : END
250 LABEL DEUX : PRINT "2": END
260 LABEL QUATRE : PRINT "4": END
```

Suivant la valeur de la variable A%, saisie au clavier, le programme effectue un branchement à l'une des adresses (numéro de ligne ou étiquette), indiquées dans la liste associée à l'instruction ON A% GOTO.

Action

Cette instruction provoque un branchement à une adresse déterminée en fonction de la partie entière de *exp-num*.

- La valeur de *exp-num* indique la position, dans la liste des adresses *adresse-n*, de celle vers laquelle s'effectue le branchement. La liste des adresses peut comprendre aussi bien des numéros de lignes que des étiquettes.
- Le cas échéant, *exp-num* est tronquée à sa partie entière. Si la valeur de celle-ci est nulle ou supérieure au nombre d'adresses indiquées, une *erreur 32* se produit ; si l'adresse sélectionnée n'existe pas, une *erreur 60* est signalée.

Aucune instruction ne doit suivre ON GOTO dans des lignes à instructions multiples.

Voir aussi : GOTO, ON GOSUB.

ON INT *exp* [*instr-comp*] — Instruction

Permet la gestion des interruptions logicielles.

exp	expression numérique entière comprise entre 0 et7.
instr-comp	Instruction(s) à exécuter.

Exemple

```
10 ON INT 7 GOTO 1000
20 N=0
100 PRINT "APPUYEZ SUR CTRL C POUR EMPECHER LE DEFILEMENT"
110 N=N+1: GOTO 100
1000 PRINT "LE MESSAGE EST APPARU";N;" FOIS"
1010 PRINT "AVANT QUE LA TOUCHE CTRL C NE SOIT PRESSEE"
```

Ce programme fait défiler en continu le même message dans la fenêtre SBASIC. tant que l'utilisateur n'appuie pas sur la touche Ctrl-C. Aussitôt que l'utilisateur appuie sur les touches Ctrl-C ou Ctrl-Break, une interruption logicielle numéro 7 est déclenchée et l'instruction GOTO 1000, placée juste derrière ON INT 7 est exécutée. Il y a donc transfert à la ligne 1000 et un message, indiquant le nombre de fois que le message « APPUYEZ SUR CTRL C TOUR EMPECHER LE DEFILEMENT » a été affiché. Enfin, l'exécution du programme s'arrête.

Action

L'instruction ON INT permet de gérer des interruptions logicielles en indiquant, pour chacune d'elles, le traitement associé.

expr-num spécifie le niveau d'interruption que l'on désire armer. Il doit, après arrondi éventuel, être compris entre 0 et 7.

- Le niveau 0 est le niveau d'interruption engendré avant l'exécution de chaque instruction du programme. Particulièrement utile en cours de mise point, il permet d'afficher le contenu de

certaines variables avant l'exécution de chaque instruction.

- Le niveau 7 correspond à l'appui sur les touches Ctrl-C ou Ctrl-Break.

instr-comp représente la ou les instructions qui seront exécutées lors de la réception d'une interruption du niveau spécifié.

- Il est possible d'effectuer, au moyen d'une instruction CALL, GOSUB ou GOTO, un branchement vers une routine de traitement de l'interruption. Une telle routine doit se terminer par une instruction RETURN.
- Lorsque *instr-comp* comporte une instruction GOTO, celle-ci doit être la dernière car les instructions placées à sa suite ne seraient pas exécutées.
- Si *instr-comp* est omise, le niveau spécifié est désarmé.

Toute nouvelle instruction ON INT annule l'effet de la précédente pour un niveau d'interruption donné.

Pendant le traitement d'une interruption, les autres interruptions sont masquées. Toute demande d'interruption est mémorisée mais n'est prise en compte qu'à la fin du traitement en cours. Toutefois, si le niveau 7 n'a pas fait l'objet d'une instruction ON INT, un appui sur Ctrl-C provoque l'arrêt du programme même si une interruption est en cours de traitement.

OPEN *nom-fich* AS *exp-num-1* [LEN *exp-num-2*] — Instruction

Ouvre un fichier à accès direct sur disque et lui affecte un canal d'entrée-sortie.

nom-fich	Nom du fichier.
exp-num-1	Numéro de canal compris entre 1 et 32.
exp-num-2	Longueur de l'enregistrement logique.

Exemple 1

```
10 OPEN "VIRT" AS 1
20 DIM #1 A(10)
30 FOR X% = 0 TO 10
40 PRINT A(X%)
50 NEXT X%
60 CLOSE 1
```

Ce programme commence par ouvrir le fichier « VIRT.DAT » (ligne 10) et l'affecte au canal 1. Ce canal est ensuite associé au tableau virtuel de réels A(*), dont tous les éléments sont affichés (lignes 30 à 50).

Exemple 2

```
10 OPEN "PERSON" AS 2 LEN 50
20 FIELD #2, 30 AS NOM$, 12 AS PRE$, 8 AS SAL$
30 FOR X% = 1 TO 10
40 GET #2 RECORD X%
50 PRINT NOM$, PRE$, CVT$F(SAL$)
60 NEXT X%
70 CLOSE 2
```

Ce programme ouvre le fichier à accès direct « PERSON.DAT » et l'affecte au canal 2 en spécifiant une longueur d'enregistrement de 50 caractères. Après la définition du tampon d'entrée-sortie (ligne 20), les dix premiers enregistrements du fichier sont lus et les trois zones qui les composent sont affichées (lignes 30 à 60) ; enfin, le fichier est fermé (ligne 70).

Action

L'instruction OPEN effectue l'ouverture du fichier à accès direct *nom-fich* et l'affecte au canal *exp-num-1* qui doit être compris entre 1 et 32.

- Le nom du fichier doit respecter les règles du système d'exploitation. Si l'extension n'est pas spécifiée, l'extension « .DAT » est ajoutée.
- Le cas échéant, la longueur de l'enregistrement est précisée par *exp-num-2*. La valeur par défaut de *exp-num-2* est 252.
- Si le fichier n'existe pas, il est automatiquement créé.

Voir aussi : OPEN NEW, OPEN OLD, OPEN APPEND, CLOSE.

OPEN { NEW | OLD | APPEND } *nom-fich* AS *exp-num* — Instruction

Ouvre un fichier séquentiel sur disque en écriture, en lecture ou en extension et lui affecte un numéro de canal.

nom-fich	Nom d'un fichier séquentiel.
exp-num	Numéro de canal compris entre 1 et 32.

Exemple 1

```
10 OPEN NEW "CARRE" AS 1
20 FOR I% = 1 TO 6
30 PRINT #1 I%*I%
40 NEXT I%
50 CLOSE 1
```

Ce programme ouvre un nouveau fichier séquentiel « CARRE.DAT » (ligne 10) sur le canal 1 puis y écrit les carrés des 6 premiers nombres entiers.

Exemple 2

```
10 OPEN OLD "CARRE" AS 1
20 FOR I% = 1 TO 6
30 INPUT #1 C%
40 PRINT C%
50 NEXT I%
60 CLOSE 1
```

Ce programme ouvre le fichier séquentiel « CARRE.DAT » en lecture (ligne 10) puis lit son contenu, à savoir les 6 premiers carrés (voir exemple 1).

Exemple 3

```
10 OPEN APPEND "CARRE" AS 1
20 FOR I% = 7 TO 12
30 PRINT #1 I%*I%
40 NEXT I%
50 CLOSE 1
```

Ce programme ouvre le fichier séquentiel « CARRE.DAT » en extension (ligne 10) puis y ajoute les carrés des nombres 7 à 12 à la suite des carrés des 6 premiers (voir exemple 1).

Action

- L'instruction OPEN NEW ouvre le fichier séquentiel *nom-fich* en écriture et lui affecte le canal *exp-num* qui doit être compris entre 1 et 32.

  Le fichier spécifié est forcément un nouveau fichier ; si un fichier de même nom existe déjà sur

le disque, il est automatiquement détruit.

- L'instruction OPEN OLD ouvre le fichier séquentiel *nom-fich* en lecture du fichier séquentiel *nom-fich* et lui affecte le canal *exp-num* qui doit être compris entre 1 et 32.
  Si le fichier spécifié n'existe pas sur le disque, une *erreur 4* est produite.
- L'instruction OPEN APPEND ouvre le fichier séquentiel *nom-fich* en écriture et prépare l'enregistrement des prochaines écritures à la fin du fichier afin de l'étendre. Si le fichier spécifié n'existe pas sur le disque, il est automatiquement créé.

Le nom du fichier doit respecter les règles du système d'exploitation. Si l'extension n'est pas spécifiée, l'extension « .DAT » est ajoutée.

Voir aussi : OPEN, CLOSE.

OPEN LIBRARY *nom-fich* AS *exp-num* — Instruction

Ouvre une librairie à chargement dynamique (DLL) et lui affecte un canal d'entrée-sortie.

nom-fich	Nom du fichier contenant la librairie.
exp-num	Numéro de canal compris entre 1 et 32.

Exemple

```
10 OPEN LIBRARY "MaDll" AS 10
20 CALL #10 "MaFonction"(X%)
30 PRINT X%
40 CLOSE 10
```

Ce programme commence par ouvrir la librairie « MaDll.DLL » et l'affecte au canal 10. La fonction « MaFonction » de la librairie « MaDll.DLL » est ensuite appelée avec la variable entière X% en paramètre. La librairie est finalement fermée.

Action

L'instruction OPEN LIBRARY effectue l'ouverture de la librairie *nom-fich* et l'affecte au canal *exp-num* qui doit être compris entre 1 et 32.

Le nom du fichier doit respecter les règles du système d'exploitation. Si l'extension n'est pas spécifiée, l'extension « .DLL » est ajoutée.

Le fichier qui contient la librairie doit être placé dans un répertoire exploré par le système d'exploitation. Sous Windows, le plus simple est de placer ce fichier dans le répertoire où se trouve le fichier « SBASIC.EXE ».

Voir aussi : CALL#, CLOSE.

OVERLAY *exp-num* — Instruction

L'instruction OVERLAY est sans effet. Elle est cependant conservée pour assurer la compatibilité avec les anciens programmes.

exp-num	Nombre compris entre 1 et 511.

Action

Aucune. Voir le paragraphe 7.2 « L'instruction OVERLAY » pour plus de détails.

PEEK(*exp-num)* — Fonction

Retourne le contenu d'un octet de la mémoire.

exp-num	Adresse mémoire.

Exemple

```
10 A = PEEK (HEX("24"))
```

Après l'exécution de cette ligne la variable réelle A contient la valeur de l'octet d'adresse hexadécimale 24.

Action

La fonction PEEK renvoie le contenu de l'octet d'adresse *exp-num*.

La valeur retournée est supérieure ou égale à 0, et inférieure ou égale à 255.

Si *exp-num* est négative ou supérieure à 2147483647, une *erreur 75* est signalée.

Voir aussi : DPEEK, POKE, DPOKE.

PI — Variable système

Variable système contenant la valeur du nombre PI.

Exemple

```
10 INPUT "RAYON ";RAYON
20 PRINT "CIRCONFERENCE = "; 2*PI*RAYON
```

Ce programme permet de calculer la circonférence d'un cercle.

Action

PI est une variable système (de même que ARGC, ARGV$, ERR, ERL, DATE$, XPEN et YPEN), contenant la valeur du nombre PI. La valeur retournée est 3.1415925535897933

PI ne peut pas figurer à gauche du signe égal (=) dans un ordre d'affectation (LET implicite ou explicite).

PLAY *nom-fich*[, *exp-logique*] — Instruction

Joue un fichier audio.

nom-fich	Nom du fichier audio.
exp-logique	Indicateur attente fin de jeu du fichier audio.

Exemple

```
10 PLAY "SON"
```

Joue le fichier « SON.WAV ».

Action

L'instruction PLAY joue le fichier audio *nom-fich* au format WAV. Si *exp-logique* est vrai, l'instruction suivante sera exécutée lorsque le fichier audio aura été joué complètement. Sinon l'exécution continue immédiatement. Si *nom-fich* est une chaîne vide, le jeu en cours est arrêté.

PLIST {[*num-ligne-1*] [,*num-ligne-2*] | *num-ligne-1*,} Commande

Liste simultanément dans la fenêtre SBASIC et dans le tampon d'impression tout ou partie du programme en mémoire.

num-ligne-1 Première ligne à lister.
num-ligne-2 Dernière ligne à lister.

Exemple 1

```
PLIST
```

Liste dans la fenêtre SBASIC et dans le tampon d'impression tout le programme.

Exemple 2

```
PLIST 10
```

Liste dans la fenêtre SBASIC et dans le tampon d'impression la ligne 10.

Exemple 3

```
PLIST 50,80
```

Liste dans la fenêtre SBASIC et dans le tampon d'impression les lignes 50 à 80.

Exemple 4

```
PLIST ,60
```

Liste dans la fenêtre SBASIC et dans le tampon d'impression de la première ligne jusqu'à la ligne 60.

Exemple 5

```
PLIST 100,
```

Liste dans la fenêtre SBASIC et dans le tampon d'impression de la ligne 100 à la dernière ligne.

Action

La commande PLIST fonctionne de même manière que la commande LIST, à cette différence que le programme est listé simultanément dans la fenêtre SBASIC et dans le tampon d'impression.

Le programme est listé de *num-ligne-1* à *num-ligne-2*. La valeur par défaut de *num-ligne-1* est 1 et celle de *num-ligne-2* est 2147483647.

Voir aussi : LIST.

+commande Commande

Permet d'exécuter une ligne de commande Windows.

commande Ligne de commande.

Exemple 1

```
+DIR *.BAS
```

La ligne de commande « DIR *.BAS » liste les fichiers ayant l'extension « .BAS » dans le répertoire courant avec affichage dans la fenêtre SBASIC.

Action

La commande + permet d'exécuter une ligne de commande. Elle ne s'utilise qu'en mode direct. Son

équivalent en mode programme est l'instruction EXEC.

Voir aussi : EXEC.

POKE *exp-num-1*, *exp-num-2* — Instruction

Écrit un octet à une adresse mémoire.

exp-num-1	Adresse mémoire.
exp-num-2	Donnée.

Exemple 1

```
10 POKE 1000,33
20 POKE L,X
30 POKE HEX('0123'),200
```

Ce programme présente quelques utilisations possibles de l'instruction POKE

Action

L'instruction POKE écrit la valeur de *exp-num-2* dans l'octet d'adresse *exp-num-1*.

exp-num-1 doit être comprise entre 0 et 2147483647.

exp-num-2 doit être comprise entre 0 et 255.

Cette instruction doit être utilisée avec beaucoup de précautions car elle peut influer sur le fonctionnement de SBASIC.

Voir aussi : DPOKE, DPEEK, PEEK.

PORT *exp-num* [, *exp-chaîne*] — Instruction

Permet d'aiguiller les informations en entrée depuis un fichier ou en sortie vers un fichier.

exp-num	Numéro de port, compris entre 0 et 19.
exp-chaîne	Chaîne de caractères contenant le nom d'un fichier.

Exemple

```
10 PRINT "ECRAN"
20 PORT 1,"FICHIER.TXT"
30 PRINT "FICHIER"
40 PORT 0
50 PRINT "ECRAN"
60 PORT 1,""
```

L'instruction PORT 1 de la ligne 20 aiguille la sortie vers le fichier « FICHIER.TXT »; de ce fait, l'ordre PRINT suivant (ligne 30) enverra son message vers ce fichier. La ligne 40 aiguille de nouveau la sortie vers la fenêtre SBASIC.

Action

L'instruction PORT permet d'aiguiller l'entrée ou la sortie vers un fichier.

exp-num est le numéro du port sélectionné ; il doit être compris entre 0 et 19.

Numéro de port	Aiguillage

0 à 9	Affichage
10 à 19	Entrée

Tableau 11 – Affectation des numéros de port

exp-chaîne est un nom de fichier.

Si *exp-chaîne* est spécifié, le port *exp-num* est associé au fichier portant ce nom.

Si *exp-chaîne* n'est pas présent, l'entrée ou la sortie est redirigé vers le fichier précédemment associé au port *exp-num*.

Si *exp-chaîne* est une chaîne vide, le fichier précédemment associé au port *exp-num* est fermé et dissocié.

Les numéros 0 (affichage écran) et 10 (entrée clavier) sont fixés une fois pour toute et ne sont pas modifiables. Ils permettent de rediriger l'entrée ou la sortie vers le clavier ou la fenêtre SBASIC.

POS(*exp-num*) — Fonction

Retourne la position courante sur le canal spécifié.

exp-num — Numéro de canal.

Exemple 1

```
10 PRINT TAB(10);"ESSAI";
20 PRINT POS(0)
```

La valeur 15 est affichée, correspondant à la position du curseur dans la fenêtre SBASIC.

Exemple 2

```
10 OPEN NEW "ESSAI.DAT" AS 1
20 PRINT #1 "ESSAI";
30 PRINT POS(1)
40 CLOSE 1
```

La valeur 5 est affichée, correspondant à la position dans le fichier ouvert sur le canal 1.

Action

La fonction POS retourne la position de colonne courante sur le canal *exp-num*.

PRINT [*exp-1*] [{ , | ; } *exp-2*] … [{ , | ; }] — Instruction

Affiche des données dans la fenêtre SBASIC.

exp-n — Expression chaîne ou numérique.

, | ; — Séparateur « , » ou « ; ».

Exemple

```
5 PRINT CHR$(12)
10 PRINT
20 PRINT "QUOI"
30 PRINT "VITESSE =";V
40 PRINT A,B;X,Y
50 PRINT "SOLUTION= "; H*G/3.2
```

Ce programme donne quelques exemples de l'utilisation possible de l'instruction PRINT :

- Envoi du code ASCII d'effacement de la fenêtre SBASIC (ligne 5),
- Saut de ligne (ligne 10),
- Affichage du mot « QUOI » (ligne 20),
- Affichage d'une constante chaîne suivie d'une variable réelle (ligne 30),
- Affichage de plusieurs variables, espacées (A et B, X et Y) ou collées (B et X) (ligne 40)
- Affichage d'une constante chaîne suivie d'une expression numérique.

Action

L'instruction PRINT permet d'afficher dans la fenêtre SBASIC une suite d'éléments pouvant être des constantes, des valeurs de variables, des expressions numériques ou de chaîne de caractères.

- Les valeurs numériques sont exprimées avec le nombre de chiffres défini par l'instruction DIGITS et sont toujours suivies d'un espace. Les valeurs positives sont précédées d'un espace, les valeurs négatives du signe moins.
- Les chaînes de caractères ne sont précédées ni suivies d'aucun espace.
- Lorsque la fin de la ligne physique est atteinte au cours d'un PRINT, le passage à la ligne suivante est automatiquement effectué.
- Si la suite d'expressions se termine par une virgule ou un point virgule, le retour à la ligne n'est pas réalisé automatiquement et la prochaine instruction PRINT affichera ses données sur la même ligne. Dans le cas contraire, un saut de ligne est effectué après l'affichage des données.

Les expressions à imprimer sont séparées soit par une virgule soit par un point virgule :

- Une expression suivie d'un point-virgule ne modifie pas la position du curseur après l'affichage de la valeur de cette expression ; la valeur suivante est donc affichée immédiatement après ;
- Une expression suivie d'une virgule provoque le saut du curseur à la position de tabulation suivante. SBASIC partage en effet chaque ligne d'édition en 5 champs de 16 caractères chacun. Ainsi, une virgule fait sauter l'affichage à la position 16, 32, 48 ou 64. A partir de la position 64, une virgule provoque un saut à la position 1 de la ligne suivante. Plusieurs virgules consécutives entraînent autant de sauts de la position d'affichage.

Remarques :

- Le mot-clé PRINT peut être remplacé par un point d'interrogation « ? ».
- Instruction PRINT sans aucun argument provoque le passage à la ligne suivante.
- Si *exp-n* est supérieur ou égale à 10^{127}, une *erreur 101* est générée.
- Si *exp-n* est inférieur ou égale à 10^{-127}, 0 est affiché.

Voir aussi : PRINT USING, LPRINT, TAB, DIGITS.

PRINT #*exp-num* [*exp-1*] [{ , | ; } *exp-2*] … [{ , | ; }] — Instruction

Permet de diriger des données vers un fichier séquentiel.

exp-num	Numéro de canal compris entre 1 et 32.
exp-n	Expression chaîne ou numérique.
, | ;	Séparateur « , » ou « ; ».

Exemple

```
10 OPEN NEW "PERSON" AS 1
20 PRINT #1 "DUPONT"
```

```
30 PRINT #1 "Jean"
40 PRINT #1, 8715.25
50 CLOSE 1
```

Les instructions PRINT #1 des lignes 20, 30 et 40 permettent d'écrire 3 données dans le fichier séquentiel « PERSON.DAT » précédemment ouvert sur le canal 1 (ligne 10).

Action

L'instruction PRINT # permet d'écrire des données dans un fichier séquentiel ouvert sur le canal *exp-num*.

Pour le reste, l'instruction PRINT # est identique à l'instruction PRINT.

Voir aussi : PRINT.

PRINT USING *exp-chaîne*, [*exp-1*] [{; | ,}*exp-2*] … [{; | ,}] — Instruction

Permet de contrôler le format d'affichage.

exp-chaîne	Chaîne de caractères décrivant le format d'affichage.
exp-n	Expression numérique ou chaîne.
;	,	« ; » ou « , »

Exemple 1

```
10 PRINT USING '! ET ! ET !' , "AX1","BCDE","C5"
```

L'exécution de cette ligne affiche dans la fenêtre SBASIC : A ET B ET C

Exemple 2

```
10 PRINT USING '\12345\ \123\',"CECI EST UN TEST","DEVOIR"
```

Affiche à l'exécution : CECI ES DEVOI

Exemple 3

```
10 PRINT USING '###.##' , 124.555
```

Affiche à l'exécution : 124.55

Exemple 4

```
10 PRINT USING '#.##' , 10.3
```

Affiche à l'exécution : %10.3

Exemple 5

```
10 PRINT USING '$$###.##' , 123.82
```

Affiche à l'exécution : $123.82

Exemple 6

```
10 PRINT USING '**###.##' , 250.30
```

Affiche à l'exécution : **250.30

Exemple 7

```
10 PRINT USING '#,###,###.##' ,1E6
```

Affiche à l'exécution : 1,000,000.00

Exemple 8

```
10 PRINT USING '$$##.## -$$##.##', 10.23, -10.23
```

Affiche à l'exécution : $10.23 -% -10.23

Exemple 9

```
10 PRINT USING '#.############^^^', SIN(1)
```

Affiche à l'exécution : 8.41470984808E-01

Action

PRINT USING est une forme très souple de 1'instruction PRINT qui donne à l'utilisateur le contrôle complet du format d'affichage.

- *exp-chaîne* est une image fidèle de la ligne d'édition ; elle comporte du texte qui sera affiché tel quel, et des **spécificateurs de format** qui seront remplacés par les valeurs des expressions de la liste. Elle doit obligatoirement se terminer par un spécificateur de format.

  Les spécificateurs de formats sont constitués des caractères de format spéciaux décrits plus loin et sont séparés par des espaces. Ils permettent de préciser sous quelle forme doivent être exprimées les valeurs des expressions de la liste.

  exp-chaîne est explorée autant de fois que nécessaire pour permettre l'affichage de tous les éléments de la liste.

- Les éléments de la liste sont du même type que ceux d'une instruction PRINT ; ce sont des expressions quelconques séparées par des virgules ou des points-virgules. Une virgule modifie la position d'affichage à la fin de chaque exploration de *exp-chaîne*, comme expliqué plus loin. Une virgule ou un point-virgule placé en fin de liste inhibe le retour au début de la ligne suivante qui est normalement effectué après l'affichage des données.

Le traitement effectué par l'instruction PRINT USING est le suivant :

1) *exp-chaîne* est explorée de gauche à droite. Une sous-chaîne de texte est affichée telle quelle.

   Lorsqu'un spécificateur de format est rencontré, l'élément suivant dans la liste *exp-n* est affiché conformément au format spécifié. En cas d'incompatibilité entre le spécificateur et le type de l'expression, une *erreur 71* se produit.

2) Lorsqu'on arrive à la fin de *exp-chaîne*, le curseur est déplacé au début de la position de tabulation suivante si le dernier élément traité est suivi d'une virgule. En effet, SBASIC partage chaque ligne d'édition en 5 champs de 16 caractères chacun. Ainsi, une virgule fait sauter l'affichage à la position 16, 32, 48 ou 64. A partir de la position 64, une virgule provoque un saut à la position 1 de la ligne suivante.

   Puis, si la liste des éléments à afficher n'est pas épuisée, l'étape 1 est réitérée.

3) Le traitement est terminé lorsque le dernier élément de la liste a été affiché. La partie non exploitée de *exp-chaîne* est ignorée. En particulier, toute sous-chaîne de texte située après le dernier spécificateur de format utilisé n'est pas affichée.

4) Si le dernier élément traité est suivi d'une virgule, ou d'un point-virgule, un retour à la ligne n'est pas effectué et le curseur conserve sa position.

Remarque : le mot-clé PRINT peut être remplacé par un point d'interrogation « ? ».

Les caractères de spécification de format sont décrits ci-après :

- **Signe Et commercial « & »**

  Le signe « & » spécifie un format chaîne de caractères. Il permet d'afficher telle quelle une expression chaîne.

- **Point d'exclamation « ! »**

  Un point d'exclamation spécifie que seul le premier d'une chaîne de caractères est à afficher.

- **Barre oblique inversée « \ »**

  Une paire de barres obliques inversées (*backslash*) encadrant n caractères quelconques définit un champ de n+2 caractères permettant d'afficher les n+2 premiers caractères d'une expression chaîne. Si la longueur de la chaîne est inférieure à n+2, elle est cadrée à gauche dans le champ et complétée par des espaces. Deux barres consécutives définissent un champ de 2 caractères.

  Les caractères placés entre les deux « \ » sont ignorés ; seul leur nombre est pris en compte. Il est conseillé d'utiliser les chiffres de 0 à 9 afin d'en faciliter visuellement le décompte.

- **Signe dièse « # »**

  Le signe dièse est utilisé pour définir un champ numérique.

  Une suite de signes dièse est utilisée pour définir un champ numérique. Chaque signe dièse représente un chiffre. Un point décimal peut être inclus dans la spécification ; il doit alors être précédé d'au moins un caractère de spécification.

  Si l'expression à afficher comporte un nombre de chiffres significatifs inférieur à la longueur du champ spécifié, elle est cadrée à droite et complétée à gauche par des espaces. La partie fractionnaire est arrondie selon le nombre de chiffres après la virgule spécifiés dans le format.

  Si la longueur du champ ne permet pas d'afficher correctement la valeur de l'expression, celle-ci est précédée d'un signe « % » puis exprimée selon le format défini par l'instruction DIGITS.

  Une valeur négative est précédée du signe moins. Ce dernier occupe une position du champ.

- **Double signe dollar « $$ »**

  Une paire de signes « $ » placée en tête d'un champ numérique permet de faire précéder la valeur affichée d'un signe dollar. Le double dollar spécifie deux positions d'affichage supplémentaires dont la première sera occupée par le signe dollar lui-même. Si la longueur du champ ne permet pas d'afficher le signe dollar, compte tenu de la valeur de l'expression, le signe % est affiché puis l'affichage est effectué comme si le double dollar était remplacé par deux dièses.

  Pour afficher avec ce format des valeurs négatives, il faut utiliser le spécificateur moins en fin de champ.

- **Double astérisque « ** »**

  Une paire de signes « * » placée en tête d'un champ numérique permet, à l'affichage, de remplacer les espaces de tête par des astérisques. Ceci est particulièrement utile lorsqu'on imprime un montant qui ne doit pas être altéré (chèque, traite etc.). Le double astérisque spécifie deux positions d'affichage supplémentaires dont la première au moins sera occupée par un astérisque.

  Si la longueur du champ ne permet pas, compte tenu de la valeur de l'expression, d'afficher au moins un astérisque, le signe % est affiché puis l'affichage est effectué comme si le double astérisque était remplacé par deux dièses.

  Pour afficher avec ce format des valeurs négatives, il faut utiliser le spécificateur moins en fin de champ.

- **Virgule « , »**

  Une ou plusieurs virgules peuvent être spécifiées afin de permettre, lors de l'affichage, l'insertion d'une virgule après le chiffre des milliers, des millions etc. Chaque virgule spécifiée constitue une position d'affichage supplémentaire ; chaque virgule insérée occupe une position d'affichage.

  La position d'une virgule à l'intérieur de la spécification est sans importance ; elle doit, toutefois, être précédée d'au moins un caractère de spécification, et apparaître à gauche du point décimal si celui-ci est aussi spécifié.

- **Signe moins « - »**

  Le signe moins « - » est utilisé pour afficher des nombres négatifs quand une paire de signes « $ » ou « * » sont placés en tête d'un champ numérique. Si le signe moins est placé en tête du champ, une valeur négative sera à l'affichage précédée de ce signe, s'il est placé à la fin du

champ numérique, une valeur négative sera suivie de ce signe.

- **Signe flèche « ^ »**
  Le signe flèche « ^ » spécifie un format scientifique dans un champs numérique. Quatre flèches, et quatre seulement, sont permises et peuvent être placées à la fin d'un champ numérique. Ces quatre flèches « ^^^^ » sont utilisées pour représenter la notation « E+XX » du format scientifique (quatre positions). Aucun autre format numérique ne peut être utilisé avec le format scientifique, car celui-ci utilise toutes les positions d'affichage. Ce format est particulièrement utile lorsqu'on affiche des tableaux de grands nombres ayant plusieurs positions décimales.

Voir aussi : LPRINT USING, PRINT, TAB, DIGITS.

PTR(*var*) Fonction

Retourne l'adresse d'une variable.

var Nom de variable.

Exemple 1

```
10 K%=200
20 P=PTR(K%)
30 PRINT PEEK(P),P      EEK(P+1),PEEK(P+2),PEEK(P+3)
```

P contient l'adresse de la variable entière K% ; la lecture directe du contenu de cette adresse donne 200, valeur de K%, pour l'octet de poids faible et 0 pour les trois autres octets de l'entier.

Exemple 2

```
10 A$="CECI EST UN ESSAI"
20 P = PTR(A$)
30 DEBUT = DPEEK(P)
40 LONGUEUR = DPEEK(P+4)
50 FOR X = DEBUT TO DEBUT+LONGUEUR-1
60 PRINT CHR$(PEEK(X))
70 NEXT X
```

Ce programme détermine l'adresse de la variable A$ ainsi que sa longueur et accède directement aux octets contenant la valeur de la chaîne. En effet, le descripteur en mémoire de la chaîne A$ dont l'adresse est retournée dans la variable P, est constitué de 8 octets : les 4 premiers octets, affectés à la variable DEBUT, pointent sur la zone de mémoire ou est stockée la chaîne de caractères proprement dite et les 4 derniers affectés à la variable LONGUEUR comportent la longueur de la chaîne.

Action

La fonction PTR retourne, selon le type d'argument, son adresse en mémoire ou celle de son descripteur.

var peut être une variable, un élément de tableau ou un tableau. Un tableau, quel que soit le nombre de ses dimensions, doit être spécifié sous la forme : *nom-du-tableau(*)*.

- Si *var* est une variable (ou un élément de tableau) numérique, l'adresse retournée est celle du premier octet ou est rangée sa valeur.
  - Les valeurs entières sont mémorisées sur 4 octets.
  - Les valeurs réelles occupent 8 octets : 52 chiffres binaires pour la mantisse, 11 pour l'exposant et 1 pour le signe.
- Si *var* est une variable (ou un élément de tableau) chaîne de caractères, l'adresse retournée est celle de son descripteur. Un descripteur de chaîne est constitué de 8 octets :

  - les 4 premiers contiennent l'adresse où commence la chaîne,
  - les 4 octets suivants contiennent la longueur de la chaîne.
- Si *var(*)* est un tableau, l'adresse retournée est celle du descripteur du tableau. Un descripteur de tableau est constitué de 6 octets :
  - Les 4 premiers contiennent l'adresse où est rangée la valeur de l'élément 0, s'il s'agit d'un tableau numérique, ou l'adresse du descripteur de l'élément 0 s'il s'agit d'un tableau chaîne de caractères. Les valeurs ou les descripteurs des éléments d'un tableau sont rangés « par lignes »,
  - Le 5e octet contient le nombre de dimensions du tableau,
  - La valeur du dernier octet indique s'il s'agit d'un tableau ordinaire (0) ou d'un tableau virtuel (1).

Voir aussi : PEEK, DPEEK, POKE, DPOKE.

PUT #*exp-num-1* [, RECORD *exp-num-2*] Instruction

Permet d'écrire un enregistrement dans un fichier à accès direct.

exp-num-1	Numéro de canal compris entre 1 et 32.
exp-num-2	Numéro d'enregistrement.

Exemple

```
10 OPEN "PERSON" AS 1 LEN 50
20 FIELD #1, 30 AS NOM$, 12 AS PRE$, 8 AS SAL$
30 INPUT "Numéro (0=fin) "; N%
40 IF N% = 0 THEN 100
50 INPUT "Nom : "; N$
60 INPUT "Prénom: "; P$
70 INPUT "Salaire: "; S
75 LSET NOM$ = N$ : LSET PRE$ = P$ : LSET SAL$ = CVTF$(S)
80 PUT #1 RECORD N%
90 GOTO 30
100 CLOSE 1
```

L'instruction PUT de la ligne 80 permet d'écrire l'enregistrement N% du fichier à accès direct « PERSON.DAT », précédemment ouvert en canal 1.

Action

L'instruction PUT permet d'écrire un enregistrement dans fichier à accès direct.

- *exp-num-1* représente le numéro de canal, compris entre 1 et 32, sur lequel le fichier à accès direct est ouvert. Toute valeur hors de ces limites provoque une *erreur 40*.
  Si aucun fichier n'est ouvert sur le canal spécifié, une *erreur 43* est signalée.
  Si le fichier n'est pas ouvert en accès direct, une *erreur 44* est provoquée.
- *exp-num-2* définit le numéro de l'enregistrement à écrire, arrondi, si nécessaire, à une valeur entière. Les enregistrements sont numérotés de 1 à 2147483647.
  Par défaut, l'enregistrement écrit est celui qui suit le dernier enregistrement lu ou écrit par une instruction GET# ou PUT#. Après une instruction OPEN, le numéro implicite est 1.

Une *erreur 7* est provoquée si l'enregistrement ne peut être écrit faute de place sur le disque.

Voir aussi : GET.

READ *var-1* [, *var-2*] … Instruction

Affecte à des variables les données associées aux instructions DATA.

var-n Nom de variable numérique ou chaîne.

Exemple

```
10 DATA JANVIER,31,FEVRIER,28,MARS,31
20 DATA AVRIL,30,MAI,31,JUIN,30
30 DATA JUILLET,31,AOUT,31,SEPTEMBRE,30
40 DATA OCTOBRE,31,NOVEMBRE,30,DECEMBRE,31
50 DIM M$(12), M%(12)
60 FOR X%=1 TO 12
70 READ M$(X%), M%(X%)
80 NEXT X%
```

Après l'exécution de ce programme, les variables M$(*) et M%(*) contiennent respectivement le nom et le nombre de jours de chacun des mois de l'année.

Action

L'instruction READ est utilisée pour lire des données dans des lignes commençant par l'instruction DATA et les affecter à des variables.

- *var-n* peut être une variable, un élément de tableau ou un élément de tableau virtuel, à l'exclusion des variables déclarées dans une instruction FIELD ou FIELD#.

  Chaque variable de la liste est séparée de la suivante par une virgule. Les données lues dans les lignes DATA sont affectées aux variables de la liste selon leur ordre d'apparition.

  Selon le nombre de variables de la liste, une instruction READ peut lire les données de plus d'une instruction DATA. Inversement, elle peut ne lire qu'une partie des données d'une instruction DATA ; l'instruction READ suivante lira alors les données restantes.

- Chaque variable doit être d'un type compatible avec celui de la donnée lue.

  Lorsqu'une donnée est affectée à une variable d'un type numérique différent, une conversion est effectuée. Dans le cas où une donnée réelle ne peut, parce qu'elle est trop grande en valeur absolue, être convertie en une valeur entière, une *erreur 104* se produit.

  Si la donnée et la variable à laquelle elle doit être affectée ne sont pas toutes deux de type numérique ou de type chaîne de caractères, une *erreur 30* est provoquée.

Une tentative de lecture au-delà de la dernière donnée provoque une *erreur 31*. L'instruction RESTORE permet, cependant, de relire des lignes DATA.

Voir aussi : DATA, RESTORE.

REFSUB Commande

Affiche ou imprime la liste des sous-programmes et des étiquettes contenus dans le programme présent en mémoire.

Exemple 1

```
REFSUB
```

La liste des sous-programmes (SUB) et des étiquettes (LABEL) est affichée.

Action

L'utilitaire REFSUB permet d'afficher ou d'imprimer la liste des sous-programmes (SUB) et des étiquettes (LABEL) apparaissant dans le programme présent en mémoire.

REM [*chaîne*] Déclaration

Introduit des remarques dans un programme.

chaîne Commentaire.

Exemple

```
10 REM ***CALCUL DE MOYENNE ***
20 SUB MOY(A(*),N%,M): LOCAL SOM, X%
30 REM Boucle: somme des N% éléments du tableau
40 FOR X%=1 TO N%
50 SOM = SOM + A(X%) : REM Mise à jour de la somme
60 NEXT X%
70 M=SOM/N% : REM M = MOYENNE
80 RETURN
```

Ce sous-programme de calcul d'une moyenne contient divers commentaires placés soit sur des lignes isolées (lignes 10 et 30) soit en fin de ligne contenant une autre instruction (lignes 50 et 70).

Action

REM est utilisé pour introduire des remarques et des commentaires à l'intérieur du programme, mais ne réalise aucune opération.

- Tout ce qui suit l'instruction REM est ignoré, y compris une instruction qui serait placée sur la même ligne.
- En revanche, les instructions précédant l'instruction REM sur la même ligne sont prises en compte.
- Il est possible de se brancher à une ligne ne contenant qu'une instruction REM.

Pour des raisons de performances, il est déconseillé d'inclure de nombreuses lignes de commentaires dans des boucles très fréquemment utilisées.

RENAME *nom-fich-1*, *nom-fich-2* Instruction

Permet de renommer un fichier disque.

nom-fich-1 Ancien nom du fichier.

nom-fich-2 Nouveau nom du fichier.

Exemple

```
10 RENAME "VIEUX", "NEUF"
```

Cette ligne renomme le fichier « VIEUX.BAS » en « NEUF.BAS ».

Action

L'instruction RENAME permet de renommer *nom-fich-1* en *nom-fich-2*. L'extension par défaut est .BAS, et le répertoire par défaut, le répertoire courant.

Si *nom-fich-1* n'existe pas, une *erreur 4* est signalée ; si *nom-fich-2* existe déjà, il se produit une *erreur 3*.

RENAME peut s'utiliser aussi bien en mode direct qu'en mode programme.

RENUM ou RENUMBER [*num-ligne-1*][,[*num-ligne-2*][,[*num-ligne-3*][,[*num-ligne-4*]]] Commande

Effectue la renumérotation totale ou partielle du programme en mémoire.

num-ligne-1	Nouveau numéro de la première ligne à numéroter.
num-ligne-2	Pas de la numérotation.
num-ligne-3	Ancien numéro de la première ligne à renuméroter.
num-ligne-4	Ancien numéro de la dernière ligne à renuméroter.

Exemple 1

```
RENUMBER
```

Le programme est renuméroté en totalité, de 10 en 10 ; le nouveau numéro de la première ligne est également 10.

Exemple 2

```
RENUMBER 100
```

Le programme est renuméroté en totalité, de 10 en 10 ; le nouveau numéro de la première ligne est 100.

Exemple 3

```
RENUM ,1,100,200
```

Le programme est renuméroté de la ligne 100 à la ligne 200 ; le nouveau numéro de la première ligne est 10 (valeur par défaut) et le pas est de 1.

Action

La commande RENUMBER (RENUM) permet de renuméroter en partie ou en totalité le programme présent en mémoire. La renumérotation affecte les lignes comprises entre *num-ligne-3* et *num-ligne-4* ; le nouveau numéro de la première ligne est *num-ligne-1* et le pas est *num-ligne-2*.

Ces arguments ont les valeurs par défaut suivantes :

num-ligne-1	10
num-ligne-2	10
num-ligne-3	1
num-ligne-4	2147483647

Comme l'indique la syntaxe, lorsque les arguments omis ne sont pas les derniers dans la liste, leur emplacement doit être marqué par des virgules tel que dans l'exemple 3.

Remarques :

- Lorsqu'un programme est renuméroté en partie, il peut se produire une tentative de chevauchement des numérotations anciennes et nouvelles. Dans ce cas, une erreur est signalée.
- Cette commande doit être utilisée avec précautions dans les programmes comportant des tests sur le contenu de la variable ERL.

RESTORE [*adresse*] Instruction

Permet à des instructions READ de relire les données des instructions DATA.

adresse	Numéro de ligne ou étiquette optionnel.

Exemple 1

```
10 DATA JANVIER,31,FEVRIER,28,MARS,31
20 DATA AVRIL,30,MAI,31,JUIN,30
30 DATA JUILLET,31,AOUT,31,SEPTEMBRE,30
40 DATA OCTOBRE,31,NOVEMBRE,30,DECEMBRE,31
50 DIM M$(12),M%(12),SEC$(6),SEC%(6)
60 FOR X%=1 TO 12: READ M$(X%),M%(X%): NEXT X%
70 RESTORE 30
80 FOR X%=1 TO 6: READ SEC$(X%),SEC%(X%): NEXT X%
```

Après l'exécution de ce programme, les tableaux M$(*) et M%(*) contiennent les caractéristiques (nom et nombre de jours) des douze mois de l'année, tandis que les six éléments des tableaux SEC$(*) et SEC%(*) contiennent celles du second semestre.

En effet, l'instruction RESTORE 30 de la ligne 50 a repositionné le pointeur de « prochaine entrée disponible » sur le premier élément de la ligne 20, de sorte que les lectures ultérieures (ligne 60) ont repris à ce niveau.

Action

L'instruction RESTORE permet à des instructions READ de relire les données des instructions DATA. La prochaine donnée disponible est alors la première donnée de l'instruction DATA dont le numéro de ligne est supérieur ou égal au numéro spécifié ou après l'étiquette.

- Si *adresse* est omise, toutes les données du programme sont à nouveau disponibles et la prochaine à être lue est la première donnée de la première instruction DATA
- Si *adresse* ne correspond à aucune ligne du programme, une *erreur 60* est signalée.

L'instruction RESTORE peut être utilisée même si les données des lignes contenant l'instruction DATA concernée n'ont jamais été lues. Il est donc possible - et prudent -, d'exécuter une instruction RESTORE au début d'un sous-programme SUB contenant des instructions DATA.

Voir aussi : DATA, READ.

RESUME { [*num-ligne*] | [NEXT] } — Instruction

Reprend le déroulement normal du programme après le traitement d'une erreur.

num-ligne	Numéro de la ligne où le programme doit reprendre.
NEXT	Reprise à l'instruction qui suit celle qui a provoqué l'erreur.

Exemple

```
10 ON ERROR GOTO 100
20 INPUT A
30 INPUT B
40 PRINT A; "divisé par "; B; "est égal à "; A/B
50 PRINT
60 GOTO 20
99 REM *** GESTION D'ERREUR ***
100 PRINT "erreur ";ERR
110 RESUME 20
```

Si une erreur survient lors de l'exécution du programme (saisie de données alphabétiques, division par 0, etc.), celui-ci est dérouté à la ligne 100. Après l'affichage d'un message signalant l'erreur, l'instruction RESUME de la ligne 110 provoque la poursuite de l'exécution à la ligne 20.

Action

L'instruction RESUME est utilisée pour reprendre le déroulement normal du programme à la fin d'une procédure de gestion d'erreur.

En effet après qu'une instruction ON ERROR ait été exécutée toute erreur provoque un déroutement vers la procédure de gestion d'erreur. Si celle-ci se termine par une instruction RESUME, l'exécution peut se poursuivre normalement.

- Si *num-ligne* est spécifiée, l'exécution se poursuit à la ligne correspondante
- Si *num-ligne* est omise ou vaut 0, le programme se poursuit à la ligne ayant provoqué l'erreur et celle-ci est ré-exécutée entièrement.
- Si NEXT est spécifié, le programme se poursuit à la ligne qui suit la ligne en erreur, sauf si l'erreur a été provoquée par une instruction de type INPUT sur une entrée au clavier. Dans ce cas, la ligne où s'est produite l'erreur est reprise en totalité.

Lorsqu'une instruction RESUME est exécutée ailleurs que dans une routine de traitement d'erreur, une *erreur 66* se produit.

Voir aussi : ON ERROR GOTO, ERR, ERL.

RETURN [CLEAR]	Instruction

Provoque le retour d'un sous-programme.

CLEAR	Retour au programme appelant de plus haut niveau.

Exemple 1

```
10 PRINT "DEBUT DU PROGRAMME PRINCIPAL"
20 GOSUB 50
30 PRINT "FIN DU PROGRAMME PRINCIPAL"
40 END
50 PRINT "SOUS-PROGRAMME"
60 RETURN
```

Le programme principal appelle (ligne 20) un sous-programme (ligne 50) qui se termine (ligne 60) en retournant au programme principal (ligne 30)

Exemple 2

```
10 PRINT "DEBUT DU PROGRAMME PRINCIPAL"
20 GOSUB 50
30 PRINT "FIN DU PROGRAMME PRINCIPAL"
40 END
50 PRINT "SOUS-PROGRAMME 1 AVANT SOUS-PROGRAMME 2"
60 GOSUB 90
70 PRINT "SOUS-PROGRAMME 1 APRES SOUS-PROGRAMME 2"
80 RETURN
90 PRINT "SOUS-PROGRAMME 2"
95 RETURN CLEAR
```

Le programme principal appelle (ligne 20) un sous-programme, qui à son tour appelle (ligne 20) un nouveau un sous-programme. Lorsqu'il se termine, ce dernier retourne directement au programme principal.

Les messages suivants s'affichent dans la fenêtre SBASIC :

```
DEBUT DU PROGRAMME PRINCIPAL
SOUS-PROGRAMME 1 AVANT SOUS-PROGRAMME 2,
SOUS-PROGRAMME 2
```

```
FIN DU PROGRAMME PRINCIPAL
```

Action

L'instruction RETURN termine l'exécution d'un sous-programme et provoque le retour au programme appelant, à la première instruction qui suit l'appel.

- Le sous-programme peut être appelé par une instruction CALL, GOSUB ou ON GOSUB.
- Dans le cas de sous-programmes s'appelant en cascade, le mot-clé CLEAR provoque un retour au programme appelant de niveau le plus élevé sans repasser par les sous-programmes intermédiaires.

Si une instruction RETURN est rencontrée sans avoir été précédée par une instruction d'appel, une *erreur 61* se produit.

Voir aussi : CALL, GOSUB, ON GOSUB.

RIGHT$(*exp-chaîne*, *exp-num*) — Fonction

Renvoie la partie droite d'une chaîne de caractères.

exp-chaîne	Chaîne de caractères.
exp-num	Nombre de caractères.

Exemple

```
10 A$ = "0123456"
20 PRINT RIGHT$(A$,3)
```

Ce programme imprime les 3 derniers caractères de la chaîne A$, à savoir, « 456 ».

Action

La fonction RIGHT$ délivre une chaîne de caractères constituée des *exp-num* caractères de droite de la chaîne de caractères *exp-chaîne*.

- Si *exp-num* est supérieure à 0 et inférieure à la longueur de la chaîne de caractères *exp-chaîne*, la partie droite de cette chaîne, composée du nombre de caractères spécifiés, est renvoyée,
- Si *exp-num* est supérieure ou égale à la longueur de la chaîne *exp-chaîne*, celle-ci est retournée en totalité,
- Si *exp-num* est nulle, une chaîne vide est renvoyée,
- Si *exp-num* est négative, une *erreur 52* est signalée,
- Si *exp-num* est supérieure à 2147483647 une *erreur 104* est signalée.

Voir aussi : LEFT$, MID$.

RND(*exp-num*) — Fonction

Permet générer un nombre aléatoire compris entre 0 et 1.

exp-num	Argument positif, négatif ou nul.

Exemple

```
10 ROUL% = 37*RND(0)
```

Après l'exécution la variable ROUL% contient un nombre compris entre 0 et 36.

Action

La fonction RND renvoie un nombre aléatoire compris entre 0 et 1. Le nombre aléatoire créé dépend de la valeur de *exp-num* :

- Si *exp-num* est négative, le même nombre aléatoire est retourné pour les mêmes valeurs de *exp-num*,
- Si *exp-num* est nulle, un nouveau nombre aléatoire est renvoyé à chaque appel de la fonction. C'est le mode d'utilisation habituel,
- Si *exp-num* est supérieure à 0, le dernier nombre engendré est répété.

Pour obtenir un nombre aléatoire A compris entre une limite inférieure incluse MIN et une limite supérieure exclue MAX, on utilise la formule suivante :

```
A = (MAX - MIN) * RND(0) + MIN
```

RSET *var-chaîne* = *exp-chaîne* — Instruction

Affecte une valeur à la partie droite d'une zone de mémoire tampon, au besoin en la tronquant ou en la complétant par des espaces.

var-chaîne	Variable chaîne de caractères.
exp-chaîne	Expression chaîne de caractères.

Exemple

```
10 FIELD 8 AS A$, 10 AS B$
20 RSET A$="12345"
30 RSET B$="CECI EST UN TEST"
```

Après l'exécution la variable A$ contient « □□□12345 » et la variable B$ contient « ST UN TEST ». En effet, la chaîne affectée à A$ étant plus courte que la longueur du tampon, elle a été précédée par des espaces ; au contraire, la chaîne affectée à B$ étant supérieure à la taille du tampon, elle a été tronquée à gauche.

Action

L'instruction RSET permet d'affecter une valeur à la partie droite zone de mémoire tampon définie par une instruction FIELD.

- *exp-chaîne* est placée dans le tampon associé à *var-chaîne* ; elle est justifiée à droite.
- Si *exp-chaîne* est plus courte que la longueur du tampon, elle est précédée par des espaces ; si elle est plus longue, elle est tronquée à gauche.

De même que les instructions SET et LSET, RSET est destinée à être utilisée avec des variables associées aux tampons d'entrée-sortie.

Voir aussi : FIELD, FIELD#, LSET, SET.

RTRIM$(*exp-chaîne*) — Fonction

Retourne une chaîne de caractères épurée des blancs de queue.

exp-chaîne	Expression chaîne de caractères.

Exemple

```
10 A$ = "□□□CECI EST UN ESSAI□□□"
20 R$ = RTRIM$(A$)
```

Après exécution de ces lignes, R$ contient « □□□CECI EST UN ESSAI »

Action

La fonction RTRIM$ renvoie le contenu de *exp-chaîne* débarrassé de ses espaces de queue.

Voir aussi : LTRIM$.

RUN [*nom-fich*] [*num-ligne*] Commande

Lance l'exécution du programme présent en mémoire ou d'un programme sauvegardé dans un fichier.

nom-fich	Nom du programme à exécuter.
num-ligne	Numéro de ligne.

Action

La commande RUN lance l'exécution du programme présent en mémoire ou d'un programme sauvegardé dans un fichier, après avoir réinitialisé toutes les variables, fermé tous les fichiers, fermé la fenêtre graphique et repositionné le pointeur des lignes DATA sur la première donnée.

nom-fich spécifie le nom sous lequel le programme à exécuter a été sauvegardé sur disque. Son extension par défaut est « BAC ». Si cet argument est omis, le programme résidant en mémoire est lancé. S'il est présent, le programme résidant initialement en mémoire est effacé et le programme *nom-fich* est chargé puis lancé.

adresse définit le numéro de la ligne où doit commencer l'exécution. Si elle est omise, l'exécution commence à la ligne portant le plus petit numéro.

Une *erreur 4* se produit si le fichier *nom-fich* n'existe pas.

Une *erreur 60* est provoquée si la ligne spécifiée n'existe pas.

La commande RUN ne peut être utilisée qu'en mode direct.

Voir aussi : CHAIN.

SAVE *nom-fich* [,*num-ligne-1*] [, *num-ligne-2*] [, *num-ligne-3*] Commande

---

Sauvegarde sous forme de texte tout ou partie d'un programme.

nom-fich	Nom du fichier dans lequel le programme est sauvegardé.
num-ligne-1	Première ligne à sauvegarder.
num-ligne-2	Dernière ligne à sauvegarder.
num-ligne-3	Translation

Exemple 1

```
SAVE "ESSAI"
```

Le programme « ESSAI.BAS » est sauvegardé en totalité, sous une forme de texte, sans modification de sa numérotation.

Exemple 2

```
SAVE "ESSAI" 1000,
```

Le programme « ESSAI.BAS » est sauvegardé de la ligne 1000 à la dernière ligne, sans modification de sa numérotation.

Exemple 3

```
SAVE "ESSAI" , ,10
```

Le programme « ESSAI.BAS » est sauvegardé du début jusqu'à la fin avec soustraction de 1à de tous les numéros de ligne.

Action

L'instruction SAVE sauvegarde sous le nom de fichier *nom-fich* et sous forme de texte la portion du programme présent en mémoire, comprise entre *num-ligne-1* et *num-ligne-2*, en retranchant de tous les numéros de lignes du programme la valeur *num-ligne-3*.

Les lignes du programme conservées dans le fichier seront donc numérotées entre *num-ligne-1-num-ligne-3* et *num-ligne-2-num-ligne-3*. Il faut évidemment que *num-ligne-3* soit inférieure à *num-ligne-1* pour que tous les numéros de lignes soient positifs.

Les valeurs par défaut de chacun des 3 paramètres optionnels sont les suivantes :

num-ligne-1	1
num-ligne-2	2147483647
num-ligne-3	0

Remarque : l'instruction SAVE efface sans avertissement tout fichier de même nom déjà présent sur le disque.

Voir aussi : COMPILE, LOAD.

SET *exp-num, var-chaîne = exp-chaîne* — Instruction

Permet de remplacer une partie d'une chaîne par une autre chaîne.

exp-num	Expression numérique entière.
var-chaîne	Nom d'une variable chaîne de caractères.
exp-chaîne	Expression chaîne de caractères.

Exemple

```
10 I% = 4
20 A$ = "123456789"
30 B$ = "--"
40 SET I%+1, A$ = B$
50 PRINT A$
```

Après l'exécution, la variable A$, dont le contenu est affiché en ligne 50, contient « 1234--789 ».

Action

L'instruction SET permet de remplacer les caractères de *var-chaîne* situés à partir du rang *exp-num* par ceux de *exp-chaîne* et cela jusqu'à épuisement de l'une des deux chaînes.

Si *exp-num* égale 1, la substitution débute au premier caractère ; si elle est supérieure à la taille de *var-chaîne*, aucune opération n'est effectuée.

Une *erreur 75* est provoquée si *exp-num* est négative ou nulle.

De même que les instructions LSET et RSET, SET est destinée à être utilisée avec des variables associées aux tampons d'entrée-sortie.

Cette instruction, qui permet de substituer une sous-chaîne de caractères dans une autre est 3 à 5 fois plus rapide que l'opération équivalente avec les instructions standard.

Voir aussi : LSET, RSET.

SETCOLOR *exp-num-1, exp-num-2, exp-num-3, exp-num-4* — Instruction

Modifie la couleur du fond et la couleur du texte affiché dans la fenêtre SBASIC.

exp-num-1	Couleur de fond (0) ou couleur du texte (1).
exp-num-2	Niveau composante rouge compris entre 0 et 256.
exp-num-3	Niveau composante verte compris entre 0 et 256.
exp-num-4	Niveau composante bleue compris entre 0 et 256.

Exemple

```
10 PRINT "    ";
20 SETCOLOR 0,255,0,0: SETCOLOR 1,0,0,255
30 PRINT "Texte en bleu sur fond rouge";
40 SETCOLOR 0,255,255,255: SETCOLOR 1,0,0,0
50 PRINT
```

Après avoir modifié la couleur de fond et la couleur du texte, ce programme affiche un texte en bleu sur fond rouge puis rétablit les couleurs d'affichage par défaut.

Action

L'instruction SETCOLOR permet de spécifier la couleur de fond et la couleur du texte affiché dans la fenêtre SBASIC.

- Si *exp-num-1* est égal à 0, c'est la couleur du fond qui est spécifiée. La valeur 1 spécifie la couleur du texte. Pour toute autre valeur, cette instruction est ignorée.
- *exp-num-2*, *exp-num-3* et *exp-num-4* définissent les niveaux des composantes rouges, vertes et bleues de la couleur spécifiée. Si celles-ci sont négatives ou supérieures à 255, une *erreur 74* est signalée.

L'instruction SETCOLOR agit sur la fenêtre texte si, ni l'instruction GR, ni l'instruction HGR, n'ont été exécutées depuis le lancement de SBASIC ou l'exécution d'une commande RUN, NEW ou CLEAR. Elle agit sur la fenêtre graphique si l'instruction TEXT n'a pas été exécutée après l'une des instructions GR et HGR. La nouvelle couleur de fond ou de texte n'est prise en compte que pour le texte affiché après l'invocation de l'instruction SETCOLOR. Elle demeure jusqu'à l'invocation d'une nouvelle instruction SETCOLOR ou l'exécution d'une commande RUN, NEW ou CLEAR.

Voir aussi : PRINT, LPRINT, RUN, NEW, CLEAR.

SGN(*exp-num*) — Fonction

Retourne le signe d'une expression numérique.

exp-num	Expression numérique quelconque.

Exemple

```
10 A = 120 : C = 0 : B = -200
20 PRINT SGN(A),SGN(B),SGN(C)
```

Ce programme affiche les valeurs 1, 0 et -1 correspondant au signe, respectivement, des variables A, B et C.

Action

La fonction SGN retourne le signe de *exp-num* de la façon suivante :

exp-num	SGN
< 0	-1
= 0	0
> 0	1

Voir aussi : ABS.

SIN(*exp-num*) Fonction

Retourne le sinus d'un angle exprimé en radians.

exp-num Angle en radians.

Exemple

```
10 PRINT SIN(PI/6)
```

La valeur retournée est égale à 0.5, correspondant au sinus de PI/6.

Action

La fonction SIN renvoie le sinus de *exp-num*.

exp-num représente un angle exprimé en radians.

Voir aussi : COS.

SPC(*exp-num*) Fonction

Permet d'envoyer d'insérer des espaces lors d'une édition.

exp-num Nombre d'espaces compris entre 0 et 255.

Exemple

```
10 PRINT "DEBUT" ; SPC(10); "SUITE"
```

Un espacement de dix caractères est généré après l'affichage du mot « DEBUT ».

Action

La fonction SPC ne peut être utilisée que dans un ordre PRINT. Elle permet d'insérer *exp-num* caractères espaces lors d'une édition.

Le cas échéant, *exp-num* est tronquée à sa partie entière. Si celle-ci est négative ou supérieure à 255, une *erreur 74* est signalée.

Voir aussi : TAB.

SQR(*exp-num*) Fonction

Retourne la racine carrée d'une expression numérique.

exp-num Expression numérique positive ou nulle.

Exemple

```
10 X=100
20 A=SQR(X)
30 PRINT A
```

Ce programme imprime la racine carrée de 100.

Action

La fonction SQR retourne la racine carrée de *exp-num*.

Si *exp-num* est négative, une *erreur 107* se produit.

STACK	Commande

Affiche la pile d'exécution du programme.

Action

La commande STACK est utilisée pour la mise au point d'un programme. Elle affiche la pile d'exécution du programme suspendue par l'instruction STOP, la frappe de Ctrl-C ou la détection d'une erreur. Elle précise les numéros de ligne des GOSUB, CALL, FOR, traitement d'erreur ou d'interruption ouverts.

Cette commande n'est utilisable qu'en mode direct.

Voir aussi : STOP.

STEP	Commande

Relance l'exécution du programme jusqu'à la ligne suivante.

Action

La commande STEP est utilisée pour la mise au point d'un programme. Elle peut être utilisée en lieu et place de la commande CONTINUE pour relancer l'exécution du programme. Cependant l'exécution s'arrête à nouveau au début de la ligne suivante. La ligne suivante est affichée. Elle sera exécutée si une nouvelle commande STEP ou une commande CONTINUE est entrée.

Cette commande n'est utilisable qu'en mode direct.

Voir aussi : CONTINUE.

STOP	Instruction

Provoque une interruption dans l'exécution du programme.

Exemple

```
10 PRINT" DEBUT "
20 STOP
30 PRINT " SUITE "
40 END
```

Ce programme est interrompu à la ligne 20. Son exécution peut être poursuivie en tapant la commande CONT.

Action

Une instruction STOP interrompt le programme en cours d'exécution et affiche un message indiquant la ligne à laquelle l'interruption s'est produite. Les variables conservent leur valeur courante et les fichiers ne sont pas fermés.

Le programme peut ensuite être relancé par la commande CONT, sauf dans les cas suivants :

- Si une erreur s'est produite au cours de l'exécution du programme.
- Si une erreur de syntaxe ou une *erreur 37* survient lors de l'entrée d'instructions en mode direct.

Voir aussi : CONT, END.

STR$(*exp-num*) Fonction

Convertit une expression numérique en une chaîne de caractères.

exp-num	Expression numérique quelconque.

Exemple

```
10 A = 123.45
20 A$ = STR$(A) + "FF"
30 PRINT A$
```

Après l'exécution, la chaîne A$ contient « □123.45□FF ».

Action

La fonction STR$ retourne une chaîne de caractères représentant la valeur de *exp-num*.

- La chaîne est identique à celle éditée par l'ordre PRINT, avec les espaces, le point décimal éventuel et le signe moins. Cette fonction utilise les paramètres définis par l'instruction DIGITS.
- Lorsque le nombre de chiffres affichés dépasse celui prévu par l'instruction DIGITS, la représentation retournée est exprimée en notation scientifique.
- Si *exp-num* est supérieur ou égale à 10^{127}, une *erreur 101* est générée.
- Si *exp-num* est inférieur ou égale à 10^{-127}, la chaîne « □0□ » est retournée.

Voir aussi : VAL.

STRING$(*exp-chaîne* , *exp-num*) Fonction

Renvoie une chaîne de caractères répétée plusieurs fois.

exp-chaîne	Chaîne de caractères.
exp-num	Expression numérique entière.

Exemple 1

```
10 TRAIT$ = STRING$("-",80)
```

Après exécution de cette ligne, la variable TRAIT$ se compose de 80 tirets (-).

Exemple 2

```
10 A$ = "01"
20 B$ = STRING$(A$+"2",5)
```

Après l'exécution du programme la variable B$ contient 5 fois la chaîne « 012 ».

Action

La fonction STRING$ renvoie *exp-num* fois le contenu de *exp-chaîne*.

- Si *exp-num* est négative, une *erreur 52* est signalée.
- Si *exp-chaîne* est vide ou si *exp-num* est nulle, une chaîne vide est renvoyée.

SUB *nom-sprog* [(*var-1*, *var-2* …)] Déclaration

Définit un sous-programme qui peut être appelé par une instruction CALL

nom-sprog	Nom du sous-programme.
var-n	Paramètre formel.

Exemple

```
110 INPUT "Nombre d'éléments ";NE%
120 DIM BB(NE%)
130 FOR X% = 1 TO NE%
140 PRINT "élément ";X%;
150 INPUT BB(X%)
160 NEXT X%
170 CALL SOMME(BB(*),NE%,SOM)
180 PRINT "La somme des ";NE%;" éléments introduits est égale à : ";SOM
190 END
…
210 SUB SOMME(A(*),N%,S)
220 S = 0
230 FOR X% = 1 TO N%
240 S = S + A(X%)
250 NEXT X%
260 RETURN
```

Ce programme lit un certain nombre de valeur numérique et les range dans un tableau BB. Puis, il appelle le sous-programme SOMME en lui transmettant dans une liste de paramètres d'appel le tableau BB, le nombre de valeurs qu'il comporte (NE%) et enfin la variable (SOM) dans laquelle le résultat doit être retourné.

Le sous-programme SOMME manipule les éléments communiqués par le programme appelant au moyen des paramètres formels A(*), N% et S représentant respectivement le tableau BB, et les variables NE% et SOM. Ces variables qui se correspondent sont de même type, mais ne portent pas nécessairement le même nom. Le sous-programme SOMME calcule la somme des N% éléments du tableau A(*) et l'affecte au paramètre formel S. L'instruction RETURN permet de reprendre le déroulement du programme appelant à la suite de l'instruction CALL. La somme calculée est alors disponible dans la variable SOM.

Action

L'instruction SUB elle permet de définir un sous-programme qui peut être appelé par une instruction CALL. Cette instruction est spécifique au SBASIC.

- (*var-1*, *var-2* …) est la liste des paramètres formels du sous-programme. Cette liste doit contenir le même nombre d'éléments que la liste de paramètres d'appels présente dans l'instruction CALL.
- Un paramètre formel peut-être une variable, un élément de tableau ou un tableau, à l'exclusion

des expressions, tableaux virtuels ou des éléments de tableaux virtuels.

- Chaque paramètre formel correspond au paramètre d'appel de même rang dans l'instruction CALL, et doit être de même type (entier, réel ou chaîne de caractères) et de même nature (variable ou tableau). A défaut, une *erreur 99* est signalée. Toutefois, il est possible de remplacer une variable par un élément d'un tableau.
- Un paramètre formel et le paramètre d'appel correspondant ne portent pas nécessairement le même nom.
- Un tableau, quel que soit le nombre de ses dimensions, doit être noté : *nom-du-tableau*(*). Il doit être dimensionné à l'intérieur du sous-programme, si et seulement si, il ne l'a pas été dans le programme appelant.
- Un paramètre formel n'a pas d'existence réelle ; il représente, à l'intérieur du sous-programme, le paramètre d'appel auquel il correspond. Toute référence à un paramètre formel est, en fait, une référence au paramètre d'appel associé.

Un sous-programme doit se terminer par une instruction RETURN qui permet de retourner au programme principal.

Voir aussi : CALL, RETURN.

SWAP *var-1*, *var-2* — Instruction

Permet d'échanger le contenu de deux variables de même type.

var-1	Variable numérique ou chaîne.
var-2	Variable numérique ou chaîne.

Exemple

```
10 IF A<B THEN SWAP A,B
```

Si la valeur de A est inférieure à celle de B, le contenu des deux variables est échangé.

Action

L'instruction SWAP échange les valeurs de *var-1* et de *var-2*. Les deux variables doivent être de même type et ne peuvent pas être des éléments de tableau virtuel.

Son intérêt est de gagner du temps lors des tris (20 à 30% de gain). Elle est particulièrement avantageuse lors du tri de chaînes de caractères, car elle provoque uniquement l'échange des pointeurs les repérant. C'est la raison pour laquelle SWAP ne permet pas d'échanger des éléments de tableau virtuel.

SYSTEM — Commande

Permet de quitter SBASIC.

Exemple

```
SYSTEM
```

Dès la validation de cette commande, on quitte le SBASIC.

Action

La commande SYSTEM permet de quitter le SBASIC. Elle n'est utilisable qu'en mode direct.

Voir aussi : EXEC, EXIT.

TAB(*exp-num*) — Fonction

Déplace la position d'affichage ou d'impression à un emplacement spécifié.

exp-num Position (entre 0 et 255).

Exemple

```
10 PRINT TAB(10); "DEBUT"; TAB(20); "SUITE"; TAB(40); "FIN"
```

Les mots « DEBUT », « SUITE » et « FIN » s'affichent respectivement à partir des colonnes 10, 20 et 40 de la fenêtre SBASIC.

Action

La fonction TAB déplace la position d'affichage ou d'impression à l'emplacement indiqué par *exp-num*.

Le cas échéant, *exp-num* est tronquée à sa partie entière. Si celle-ci est négative ou supérieure à 255, une *erreur 74* est signalée.

Cette fonction ne permet pas de déplacer la position d'affichage en arrière sur la même ligne.

La fonction TAB ne peut être utilisée que dans un ordre PRINT

Voir aussi : SPC.

TAN(*exp-num*) — Fonction

Retourne la tangente d'un angle exprimé en radians.

exp-num Angle en radians.

Exemple

```
10 PRINT 1/TAN(3*PI/2)
```

La valeur retournée est égale à 0, correspondant à l divisé par l'infini.

Action

La fonction TAN renvoie la tangente de *exp-num*.

exp-num représente un angle, exprimé en radians.

Voir aussi : ATN.

TASVAR — Commande

Met à jour la table des symboles.

Exemple

```
TASVAR
```

Après l'exécution de cette commande, la table des symboles ne comprend plus les symboles inutiles ou générés par des fautes de frappe.

Action

La commande TASVAR permet de mettre à jour la table des symboles en supprimant les symboles inutiles ou résultant de fautes de frappe lors de l'introduction de lignes de programme ou d'instructions en mode direct. TASVAR est notamment utilisé avant d'effectuer un tri des variables au moyen de la commande TRIVAR.

Voir aussi : TRIVAR.

TEXT [*exp-num-1, exp-num-2]* Instruction

Définit le nombre de lignes et de colonnes de la fenêtre SBASIC.

exp-num-1	Nombre de lignes de la fenêtre SBASIC (80 à 255).
exp-num-2	Nombre de colonnes de la fenêtre SBASIC (25 à 255).

Exemple

```
10 TEXT 50,200
```

Ce programme redimensionne la fenêtre SBASIC avec 50 lignes de 200 caractères.

Action

Si *exp-num-1* et *exp-num-2* ne sont pas spécifiés ou s'ils sont égaux aux dimensions courantes de la fenêtre SBASIC, l'instruction TEXT met en avant plan cette fenêtre.

Sinon la fenêtre SBASIC est redimensionnée et effacée. *exp-num-1* doit être compris entre 25 et 255. *exp-num-2* doit être compris entre 80 et 255. Au besoin *exp-num-1* et *exp-num-2* sont tronqués. Les effets des instructions WINDOW et SETCOLOR sur cette fenêtre SBASIC sont annulés.

Voir aussi : WINDOW, SETCOLOR.

TIME [{PRINT|*var-num*}] Instruction

Permet de mesurer le temps d'exécution d'intructions.

var-num	Variable dans laquelle est stockée le temps écoulé.

Exemple 1

```
10 TIME
20 FOR I = 1 TO 1000000000
30 NEXT I
40 TIME PRINT
```

La ligne 20 met à zéro le compteur de temps écoulé. La ligne 40 affiche le temps écoulé depuis la dernière exécution de l'instruction TIME sans argument. Le résultat affiché indique le temps nécessaire à l'exécution d'un milliard de boucles vides.

Exemple 2

```
10 I = 0
20 TIME
30 I = I + 1
40 TIME T
50 IF T < 60 GOTO 30
60 PRINT "Nombre de boucles exécutées en 1 minute: ";I
```

Action

- L'instruction TIME sans argument met à zéro le compteur de temps écoulé.
- TIME PRINT affiche le temps écoulé depuis la dernière exécution de l'instruction TIME sans argument.
- TIME *var-num* affecte à la variable numérique le temps écoulé en secondes depuis la dernière exécution de l'instruction TIME sans argument.
- TRIVAR Commande
- Affiche la table des références croisées du programme.
-
- Exemple

TRIVAR	Commande

La table des symboles s'affiche dans la fenêtre SBASIC.

Action

L'instruction TRIVAR permet d'afficher la table des symboles du programme, triée dans l'ordre alphabétique : noms de variables, noms de sous-programmes, noms de tableaux et étiquettes existant dans le programme. Chaque nom est suivi de la liste des numéros des lignes dans lesquelles il apparaît : ce sont les références croisées du programme.

Voir aussi : TASVAR.

TROFF ou TRACE OFF	Instruction

Désactive la fonction TRACE.

Exemple

```
10 PRINT " DEBUT "
20 TRON
30 PRINT " PHASE II "
40 PRINT " PHASE III "
50 TROFF
60 PRINT " PHASE IV "
70 END
```

L'instruction TROFF de la ligne 50 met fin à la fonction TRACE activée à la ligne 20.

Action

L'instruction TROFF (TRACE OFF) désactive la fonction *trace* mise en œuvre par l'instruction TRON. Elle s'utilise aussi bien en mode programme qu'en mode direct.

La notation alternative TRACE OFF n'est utilisable qu'en mode direct.

Voir aussi : TRON.

TRON [*instr-comp*] ou TRACE ON	Instruction

Active la fonction *trace*, en exécutant optionnellement une série d'instructions à chaque ligne tracée.

instr-comp Instruction composée.

Exemple 1

```
10 REM DEBUT
20 TRON
30 PRINT " SUITE "
40 GOTO 60
50 REM *** LE PROGRAMME NE PASSE PAS ICI ***
60 END
```

A partir de la ligne 20, tous les numéros des lignes par lesquelles le programme passe seront affichés.

Exemple 2

```
10 TRON PRINT "A=";A;"B=";B; : C=C+1 : PRINT "Trace:";C
20 INPUT " A ET B"; A,B
30 A=A*2: B=B*3
40 A=A+50: B=B+120
50 END
```

La ligne 10 active la fonction TRACE tout en indiquant une série d'instructions qui devront être exécutées à chaque fois qu'un numéro de ligne est affiché. Ceci permet de suivre l'évolution des variables A et B ; la variable C est incrémentée puis affichée à chaque exécution d'une nouvelle ligne.

Action

- L'instruction TRON (TRACE ON) permet de faire fonctionner le programme en mode *trace* : les numéros de ligne de chaque instruction exécutée sont affichés au fur et à mesure du déroulement du programme. L'instruction peut être utilisée soit en mode direct, avant les commandes RUN ou CONTINUE, soit en mode programme.
- Lorsqu'elle est utilisée en mode programme, l'instruction TRON peut être suivie d'une ou de plusieurs d'instructions SBASIC qui seront exécutées à chaque passage par une nouvelle ligne tant que la fonction TRACE est active. Cette possibilité constitue un puissant moyen de mise au point de programmes, en affichant, par exemple, les valeurs de variables critiques.

La fonction TRACE est désactivée par l'instruction TROFF ou par la commande NEW.

La notation alternative TRACE ON n'est utilisable qu'en mode direct.

Voir aussi : TROFF.

VAL(*exp-chaîne*) Fonction

Convertit une chaîne de caractères numériques en la valeur correspondante.

exp-chaîne Chaîne de caractères numérique.

Exemple

```
10 A$ = "123"
20 A = VAL (A$)
30 B = VAL (A$+".25")
40 PRINT A,B
```

Après exécution, les variables A et B contiennent respectivement les valeurs 123 et 123.25.

Action

La fonction VAL est l'inverse de la fonction STR$. Elle convertit *exp-chaîne* en la valeur numérique correspondante.

- *exp-chaîne* est une chaîne de caractère représentant un nombre entier ou réel dont la valeur est comprise entre 10^{-127} et 10^{+127}.
- L'évaluation de *exp-chaîne* s'arrête au premier symbole ne pouvant s'interpréter comme faisant partie du nombre analysé.

Voir aussi : STR$.

WINDOW *exp-num-1*, *exp-num-2*, *exp-num-3*, *exp-num-4* — Instruction

Définit la sous-fenêtre rectangulaire d'affichage dans la fenêtre SBASIC.

exp-num-1	Ligne du coin supérieur gauche.
exp-num-2	Colonne du coin supérieur gauche.
exp-num-3	Ligne du coin inférieur droit.
exp-num-4	Colonne du coin inférieur droit.

Exemple 1

```
WINDOW 5,20,15,60
```

L'affichage du texte dans la fenêtre SBASIC s'effectue dans un rectangle délimité par les lignes 5 et 15 et par les colonnes 20 et 60.

Exemple 2

```
WINDOW 0,1,24,80
```

L'affichage se fait du texte utilise l'intégralité de la fenêtre SBASIC.

Action

L'instruction WINDOW permet de définir une sous-fenêtre rectangulaire d'affichage du texte dans la fenêtre SBASIC.

(*exp-num-1*, *exp-num-2*) et (*exp-num-3*, *exp-num-4*) sont la ligne et la colonne de deux points diagonaux dans la fenêtre SBASIC. La valeur de chaque expression doit être comprise entre -2147483648 et 2147483647. Si les valeurs dépassant les limites de la fenêtre SBASIC, à savoir 0-24 pour les lignes et 1-80 pour les colonnes, elles sont ramenées à ces limites. Notez que les numéros de ligne et de colonne du caractère en haut à gauche de la fenêtre SBASIC sont respectivement 0 et 1.

Si la fenêtre SBASIC est redimensionnée par l'instruction TEXT, les valeurs maximales de *exp-num-1* et *exp-num-2* évoluent en conséquence.

L'instruction WINDOW permet de travailler en mode multifenêtre, à la fois en mode graphique et en mode texte, en redéfinissant une fenêtre chaque fois qu'on l'utilise. Elle agit sur la fenêtre graphique si l'instruction TEXT n'a pas été exécutée après l'une des instructions GR et HGR. Elle agit sur la fenêtre SBASIC si, ni l'instruction GR, ni l'instruction HGR, n'ont été exécutées depuis le lancement de SBASIC ou l'exécution d'une commande RUN, NEW ou CLEAR.

L'instruction WINDOW s'applique aux ordres CLS et CURSOR ainsi qu'aux variables système XPEN et YPEN. Si le curseur est hors de la fenêtre lors de sa définition, il est positionné sur le bord le plus proche.

Les valeurs XPEN et YPEN sont données par rapport aux bords de la fenêtre.

Voir aussi : CLS, CURSOR, XPEN, YPEN.

XPEN — Variable système

Variable système contenant l'abscisse du curseur dans la fenêtre SBASIC après l'exécution de l'instruction CURSOR.

Exemple

```
10 CURSOR
20 IF (XPEN>= 10 AND XPEN <=21) THEN 40
30 PRINT "Le curseur n'est pas entre les colonnes 10 et 21."
35 END
40 PRINT "Le curseur est entre les colonnes 10 et 21."
45 END
```

La ligne 10 lit la position du curseur dans la fenêtre SBASIC et affecte sa position horizontale à XPEN et sa position verticale à YPEN.

Action

XPEN est une variable système (de même que YPEN, ARGC, ARGV$, ERR, ERL, PI ou DATE$) donnant la position du curseur sur 1'axe horizontal lors du dernier appel de la fonction CURSOR. La valeur par défaut de XPEN est la largeur maximale de la fenêtre graphique.

La variable XPEN peut être utilisée dans des expressions comme toute autre variable numérique, mais elle ne peut pas figurer à gauche du signe égal (=) dans un ordre d'affectation (LET explicite ou implicite).

Voir aussi : CURSOR, YPEN.

YPEN — Variable système

Variable système contenant l'ordonnée du curseur dans la fenêtre SBASIC après l'exécution de l'instruction CURSOR

Exemple

```
10 CURSOR
20 IF (YPEN>= 10 AND YPEN <=15) THEN 40
30 PRINT "Le curseur n'est pas entre les lignes 10 et 15."
35 END
40 PRINT "Le curseur est entre les lignes 10 et 15."
45 END
```

La ligne 10 lit la position du curseur dans la fenêtre SBASIC et affecte sa position horizontale à XPEN et sa position verticale à YPEN.

Action

YPEN est une variable système (de même que XPEN, ARGC, ARGV$, ERR, ERL, PI ou DATE$) donnant la position du curseur sur 1'axe vertical lors du dernier appel de la fonction CURSOR. La valeur par défaut de YPEN est la hauteur maximale de la fenêtre graphique.

La variable YPEN peut être utilisée dans des expressions comme toute autre variable numérique, mais elle ne peut pas figurer à gauche du signe égal (=) dans un ordre d'affectation (LET explicite ou implicite).

Voir aussi : CURSOR, XPEN.

Troisième partie : Instructions graphiques

Les instructions graphiques permettent de réaliser des dessins dans une fenêtre graphique. Cette fenêtre doit être préalablement ouverte par l'instruction GR ou HGR. Elle se ferme lorsqu'une instruction NEW, RUN ou CLEAR sans argument est exécutée.

Le point de coordonnée (0,0) est le coin en bas à gauche. Le point en haut à droite a pour coordonnées (319,199) en mode GR et (799,599) en mode HGR. Le premier nombre représente la position sur l'axe horizontal et le second la position sur l'axe vertical.

Par défaut l'instruction GR ouvre une fenêtre graphique de 320 par 200. Cependant il est possible spécifier la largeur et la hauteur de la fenêtre graphique lors de sa création.

Les instructions graphiques agissent sur un écran virtuel où les coordonnées horizontales et verticales peuvent varier entre -2147483648 et +2147483647. Seule la partie correspondant à la fenêtre graphique est affichée.

ARC *exp-num-1*, *exp-num-2* [,*exp-num-3*] [,*exp-num-4*, *exp-num-5*] Instruction

---

Trace un arc de cercle ou un cercle complet dans la fenêtre graphique.

exp-num-1	: Abscisse du centre.
exp-num-2	: Ordonnée du centre.
exp-num-3	: Angle, en radians.
exp-num-4	: Abscisse du point de départ.
exp-num-5	: Ordonnée du point de départ.

Exemple

```
10 HGR
20 COLOR 0
30 CLRG
40 COLOR 15
50 ARC 200,200,PI,40,40
60 ARC 400,400,2*PI,500,500
```

La ligne 50 permet de tracer un demi-cercle ayant pour centre le point (200,200) et pour départ du tracé le point (40,40) ; la ligne suivante (60) provoque l'affichage d'un cercle complet ayant pour centre 400,400 et pour départ du tracé le point 500,500.

Action

L'instruction ARC permet de tracer dans la fenêtre graphique un arc de cercle ou un cercle complet.

- (*exp-num-1*, *exp-num-2*) sont les coordonnées du centre. Elles peuvent être comprises entre -2147483648 et 2147483647 ; il s'agit d'un point virtuel éventuellement situé hors de 1a fenêtre graphique.
- *exp-num-3* représente l'angle de l'arc exprimé en radians. Si sa valeur est omise, si elle est supérieure à 2*PI ou inférieure à -2*PI, le cercle complet est tracé. Si l'angle est positif, le sens du tracé est le sens trigonométrique direct (c'est-à-dire le sens contraire des aiguilles d'une montre). Si l'angle est négatif le sens du tracé est le sens trigonométrique inverse.
- (*exp-num-4*, *exp-num-5*) sont les coordonnées du point de départ du tracé. Si elles sont omises, le tracé commence au point courant.

CLRG — Instruction

Efface la fenêtre graphique dans la couleur courante.

Exemple

```
10 HGR
20 COLOR 15
30 CLRG
```

Ce programme sélectionne la couleur 15 comme couleur courante et illumine uniformément la fenêtre graphique dans cette couleur.

Action

L'instruction CLRG effectue un effacement complet de la fenêtre graphique dans la couleur courante.

Voir aussi : COLOR.

COLOR *exp-num* — Instruction

Sélectionne la couleur logique courante.

exp-num Numéro de couleur logique compris entre 0 et 255.

Exemple

```
10 HGR
20 COLOR 12
30 PLOT 100,100 TO 500,500
```

La diagonale tracée entre le point de coordonnées (100,100) et le point de coordonnées (500,500) apparaît dans la couleur 12 (Rouge clair).

Action

L'instruction COLOR sélectionne la couleur logique courante, c'est-à-dire celle dans laquelle seront effectuées toutes les opérations graphiques suivantes (PLOT, CLRG, …), jusqu'à une nouvelle modification de la couleur courante.

exp-num définit la couleur choisie dans la palette graphique. Si celle-ci est négative ou supérieure à 255, une *erreur 74* est signalée.

Par défaut la palette graphique est la palette VGA. Elle peut être modifiée par l'instruction

SETCOLOR.

Les couleurs de 0 à 15 de la palette VGA sont les suivantes :

exp-num	**Couleur**
0	Noir
1	Bleu
2	Vert
3	Cyan
4	Rouge
5	Magenta
6	Marron
7	Gris clair
8	Gris foncé
9	Bleu clair
10	Vert clair
11	Cyan clair
12	Rouge clair
13	Magenta clair
14	Jaune
15	Blanc

Tableau 12 – Couleurs logiques 0 à 15

Les couleurs de 16 à 31 de la palette VGA sont les niveaux de gris de noir (couleur 16) à blanc (couleur 31).

Les couleurs de 32 à 246 de la palette VGA ont les niveaux de couleurs rouge, vert et bleu suivants :

exp-num	Rouge	Vert	Bleu
32	0	255	0
33	64	255	0
34	127	255	0
35	191	255	0
36	255	255	0
37	255	191	0
38	255	127	0
39	255	64	0
40	0	0	255
41	0	64	255
42	0	127	255
43	0	191	255
44	0	255	255
45	0	255	191
46	0	255	127
47	0	255	64
48	255	0	0
49	255	0	64
50	255	0	127
51	255	0	191
52	255	0	255

exp-num	Rouge	Vert	Bleu
140	58	112	112
141	58	112	96
142	58	112	85
143	58	112	69
144	112	58	58
145	112	58	69
146	112	58	85
147	112	58	96
148	112	58	112
149	96	58	112
150	85	58	112
151	69	58	112
152	80	112	80
153	90	112	80
154	96	112	80
155	106	112	80
156	112	112	80
157	112	106	80
158	112	96	80
159	112	90	80
160	80	80	112

53	191	0	255
54	127	0	255
55	64	0	255
56	127	255	127
57	159	255	127
58	191	255	127
59	223	255	127
60	255	255	127
61	255	223	127
62	255	191	127
63	255	159	127
64	127	127	255
65	127	159	255
66	127	191	255
67	127	223	255
68	127	255	255
69	127	255	223
70	127	255	191
71	127	255	159
72	255	127	127
73	255	127	159
74	255	127	191
75	255	127	223
76	255	127	255
77	223	127	255
78	191	127	255
79	159	127	255
80	181	255	181
81	197	255	181
82	218	255	181
83	234	255	181
84	255	255	181
85	255	234	181
86	255	218	181
87	255	197	181
88	181	181	255
89	181	197	255
90	181	218	255
91	181	234	255
92	181	255	255
93	181	255	234
94	181	255	218
95	181	255	197
96	255	181	181
97	255	181	197
98	255	181	218
99	255	181	234
100	255	181	255
101	234	181	255
102	218	181	255
103	197	181	255

161	80	90	112
162	80	96	112
163	80	106	112
164	80	112	112
165	80	112	106
166	80	112	96
167	80	112	90
168	112	80	80
169	112	80	90
170	112	80	96
171	112	80	106
172	112	80	112
173	106	80	112
174	96	80	112
175	90	80	112
176	0	64	0
177	16	64	0
178	32	64	0
179	48	64	0
180	64	64	0
181	64	48	0
182	64	32	0
183	64	16	0
184	0	0	64
185	0	16	64
186	0	32	64
187	0	48	64
188	0	64	64
189	0	64	48
190	0	64	32
191	0	64	16
192	64	0	0
193	64	0	16
194	64	0	32
195	64	0	48
196	64	0	64
197	48	0	64
198	32	0	64
199	16	0	64
200	32	64	32
201	42	64	32
202	48	64	32
203	58	64	32
204	64	64	32
205	64	58	32
206	64	48	32
207	64	42	32
208	32	32	64
209	32	42	64
210	32	48	64
211	32	58	64

104	0	112	0
105	31	112	0
106	58	112	0
107	85	112	0
108	112	112	0
109	112	85	0
110	112	58	0
111	112	31	0
112	0	0	112
113	0	31	112
114	0	58	112
115	0	85	112
116	0	112	112
117	0	112	85
118	0	112	58
119	0	112	31
120	112	0	0
121	112	0	31
122	112	0	58
123	112	0	85
124	112	0	112
125	85	0	112
126	58	0	112
127	31	0	112
128	58	112	58
129	69	112	58
130	85	112	58
131	96	112	58
132	112	112	58
133	112	96	58
134	112	85	58
135	112	69	58
136	58	58	112
137	58	69	112
138	58	85	112
139	58	96	112

212	32	64	64
213	32	64	58
214	32	64	48
215	32	64	42
216	64	32	32
217	64	32	42
218	64	32	48
219	64	32	58
220	64	32	64
221	58	32	64
222	48	32	64
223	42	32	64
224	47	64	47
225	48	64	47
226	53	64	47
227	63	64	47
228	64	64	47
229	64	63	47
230	64	53	47
231	64	48	47
232	47	47	64
233	47	48	64
234	47	53	64
235	47	63	64
236	47	64	64
237	47	64	63
238	47	64	53
239	47	64	48
240	64	47	47
241	64	47	48
242	64	47	53
243	64	47	63
244	64	47	64
245	63	47	64
246	53	47	64
247	48	47	64

Tableau 13 – Couleurs logiques 32 à 247

Voir aussi : CLRG, SETCOLOR.

DASH *exp-num* — Instruction

Précise la nature du trait des tracés dans la fenêtre graphique.

exp-num Nature du tracé compris entre 0 et 3.

Exemple

```
10 HGR
20 COLOR 0
30 CLRG
```

```
40 COLOR 15
50 DASH 1
60 PLOT 10,10 TO 100,100
70 DASH 2
80 PLOT 150,150 TO 300,400
```

Le segment tracé entre les points (10,10) et (100,100) apparaît sous la forme d'un trait pointillé fin, tandis que le segment (150,150) à (300,400) apparaîtra sous la forme d'un pointillé fort.

Action

L'instruction DASH permet de préciser la nature et la largeur du trait du tracé, en fonction de la valeur de *exp-num*.

exp-num	**Nature du trait**
0	Trait continu
1	Pointillé fin
2	Pointillé fort
3	Mixte
4	Trait continu avec inversion d'allumage

Tableau 14 – Sélection de la nature du trait dans l'instruction DASH

Si différent de 4, *exp-num* est tronquée à sa partie entière modulo 3. Si celle-ci est négative ou supérieure à 255, une *erreur 74* est signalée.

La valeur définissant la nature du trait reste inchangée entre deux instructions DASH.

DRAW *exp-num-1*, *exp-num-2*, *exp-chaîne* [,*exp-num-3*] — Instruction

Dessine de petits motifs répétitifs.

exp-num-1	Abscisse du point de départ.
exp-num-2	Ordonnée du point de départ.
exp-chaîne	Chaîne de caractères définissant couleurs et déplacements relatif.
exp-num-3	Rotation en multiples de 45 degrés.

Exemple

```
10 HGR
20 COLOR 0
30 CLRG
40 A$="3Xn2Nx3Dn2Nd"
50 DRAW 130,130,A$
60 DRAW 200,200,A$,1
70 DRAW 50,50,A$
```

Ce programme définit un petit rectangle dont deux côtés sont en jaune et deux autres en vert (variable A$). Cette figure est tracée en trois endroits de la fenêtre graphique. Elle subit une rotation de 45 degrés lorsqu'elle est dessinée au point de coordonnées 200,200 (ligne 60).

Action

L'instruction DRAW permet de dessiner de petits motifs répétitifs dans la fenêtre graphique.

- Le tracé commence au point (*exp-num-1*, *exp-num-2*).
- *exp-chaîne* définit le tracé du dessin par une succession de segments de droite. Ceux-ci sont décrits par des chiffres et des lettres majuscules ou minuscules ayant la signification suivante :
  - Chiffre
    0 à 7 définit la couleur logique du tracé
    8 supprime le tracé
    Par défaut, la couleur courante est utilisée.
  - Majuscule de A à Z.
    Définit le déplacement relatif en X par rapport au point courant (voir tableau de correspondances).
  - Minuscule de a à z.
    Définit le déplacement relatif en Y par rapport au point courant (voir tableau de correspondances).
- *exp-num-3* représente la rotation. Sa valeur est comprise entre 0 et 7. La rotation effectuée correspond à *exp-num* x 45 degrés par rapport au point d'origine. Par défaut la rotation est nulle.

Correspondance entre lettres et déplacements relatifs :

A	B	C	D	E	F	G	H	I	J	K	L	M
-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

N	0	P	Q	R	S	T	U	V	W	X	Y	Z
0	1	2	3	4	5	6	7	8	9	10	11	12

Tableau 15 – Codes des déplacements relatifs dans l'instruction DRAW

Correspondance entre chiffre et angle de rotation en degrés :

0	1	2	3	4	5	6	7
0	45	90	135	180	225	270	315

Tableau 16 – Codes des angles de rotation dans l'instruction DRAW

FILL *exp-num-1*, *exp-num-2* [,*exp-num-3*] — Instruction

Permet de remplir une zone de la fenêtre graphique avec la couleur logique courante.

exp-num-1	Abscisse d'un point.
exp-num-2	Ordonnée d'un point.
exp-num-3	Couleur.

Exemple

```
100 HGR
110 COLOR 0
120 CLRG
130 COLOR 15
200 PLOT 100,100: REM origine
209 REM *** tracé d'un rectangle ***
210 PLOT TO 160,100
220 PLOT TO 160,160
230 PLOT TO 100,160
```

```
240 PLOT TO 100,100
299 REM *** tracé d'un cercle ***
300 COLOR 2
310 ARC 130,130,2*PI,140,140
315 REM *** le cercle est "rempli" avec la couleur 2 ***
320 FILL 130,130
```

Ce programme trace un rectangle (lignes 210 à 240) à l'intérieur duquel un petit cercle est ensuite dessiné (ligne 310). Enfin, l'instruction FILL de la ligne 320 permet de remplir ce petit cercle de la couleur courante définie en ligne 300.

Action

L'instruction FILL permet de remplir de la couleur courante la zone colorée à laquelle appartient le point de coordonnées (*exp-num-1*, *exp-num-2*). La zone colorée peut être éventuellement délimitée par la couleur *exp-num-3*.

GGET *var-chaîne*, *exp-num-1*, *exp-num-2*, *exp-num-3*, *exp-num-4* — Instruction

Range, dans une variable chaîne, l'image bitmap d'un rectangle de la fenêtre graphique.

var-chaîne	Variable chaîne recevant la bitmap.
exp-num-1	Abscisse du coin inférieur gauche.
exp-num-2	Ordonnée du coin inférieur gauche.
exp-num-3	Abscisse du point supérieur droit.
exp-num-4	Ordonnée du supérieur droit.

Exemple

```
10 GGET B$,100,50,109,59
20 BSAVE "ICONE.BMP" B$
```

Ce programme crée le fichier ICONE.BMP contenant l'image bitmap du rectangle de 10 x 10 pixels dont l'angle inférieur gauche se trouve à l'abscisse 100 et l'ordonnée 50 de la fenêtre graphique.

Action

L'instruction GGET permet de générer l'image bitmap du contenu d'un rectangle de la fenêtre graphique. *exp-num-1* et *exp-num-2* sont les coordonnées du coin inférieur gauche du rectangle. *exp-num-3* et *exp-num-4* sont les coordonnées du coin supérieur droit. L'image bitmap est rangée dans la variable chaîne *var-chaîne*. Elle est au format BMP 256 couleurs.

Les parties du rectangle extérieures à la fenêtre graphique prennent la couleur 0.

Voir aussi : GPUT, BSAVE.

GPUT *var-chaîne*, *exp-num-1*, *exp-num-2* [, *exp-num-3* [, *exp-num4*, *exp-num5*, *exp-num6*, *exp-num7*]]

Affiche dans la fenêtre graphique l'image bitmap contenue dans une variable chaîne.

var-chaîne	Variable chaîne contenant la bitmap.
exp-num-1	Abscisse du coin inférieur gauche dans la fenêtre graphique.
exp-num-2	Ordonnée du coin inférieur gauche dans la fenêtre graphique.
exp-num-3	Mode traitement couleur.
exp-num-4	Abscisse du coin inférieur gauche dans l'image.
exp-num-5	Ordonnée du coin inférieur gauche dans l'image.
exp-num-6	Abscisse du coin supérieur droit dans l'image.

exp-num-7 Ordonnée du coin supérieur droit dans l'image.

Exemple 1

```
10 GGET B$,100,40,109,59
20 GPUT B$,50,40
```

Ce programme recopie le contenu du rectangle de 10 x 20 pixels dont l'angle inférieur gauche se trouve à l'abscisse 100 et l'ordonnée 40 de la fenêtre graphique dans le rectangle de même taille dont l'angle inférieur gauche se trouve à l'abscisse 50 et l'ordonnée 40.

Exemple 2

```
100 HGR
110 BLOAD "IMAGE.BMP" I$
120 GPUT I$,0,0
130 H = 35 : W = 25
140 PEN
150 GGET BM$, XPEN - W, YPEN - H, XPEN + W - 1, YPEN + H - 1
160 GGET BMSAVE$, XPEN - W, YPEN - H, XPEN + W - 1, YPEN + H - 1
170 GPUT BM$, XPEN - W, YPEN - H, 1
180 XCUR = XPEN : YCUR = YPEN
190 PEN 2, A$
200 IF A$=' ' THEN GOTO 230 ELSE IF XCUR = XPEN AND YCUR = YPEN GOTO 190
210 GPUT BMSAVE$, XCUR - W , YCUR - H
220 GOTO 160
230 TEXT
240 END
```

Ce programme affiche une image. Suite à un clic de souris, le rectangle de 25 x 35 pixels, centré sur la position de la souris, suit le déplacement de la souris jusqu'à ce que la barre d'espace soit appuyée.

Action

L'instruction GPUT permet d'afficher une image bitmap dans la fenêtre graphique. *exp-num-1* et *exp-num-2* sont les coordonnées du coin inférieur gauche du rectangle affiché. L'image bitmap est prise à partir de la variable chaîne *var-chaîne*. Cette image doit être au format BMP 256 couleurs. La palette de couleurs de l'image est ignorée.

Si *exp-num-3* est spécifié, il définit le mode de traitement de la couleur.

exp-num-3	Traitement couleur
0	Pas de traitement
1	Ignore la couleur de fond de l'image bitmap
2	Inversion couleur (XOR)

Tableau 17 – Mode de traitement de la couleur dans l'instruction GPUT

Optionnellement *exp-num-4*, *exp-num-5*, *exp-num-6* et *exp-num-7* définissent un rectangle à l'intérieur de l'image bitmap. C'est alors seulement l'image contenue dans ce rectangle qui est recopiée dans la fenêtre graphique. Si *exp-num-6* ou *exp-num-7* est négatif, l'abscisse ou l'ordonnée du coin supérieur droit du rectangle est définie relativement au coin supérieur droit de l'image bitmap.

Les parties de l'image bitmap qui sortent de la fenêtre graphique sont ignorées et aucune erreur n'est générée.

Voir aussi : GGET, BLOAD.

GR [*exp-num-1*, *exp-num-2]* — Instruction

Initialise la fenêtre graphique.

exp-num-1	Largeur de la fenêtre graphique.
exp-num-2	Hauteur de la fenêtre graphique.

Exemple 1

```
10 GR
20 PLOT 10,10 TO 100,100
```

Ce programme initialise ouvre une fenêtre graphique de 320 par 200 points et trace une droite du point (10,10) au point (100,100).

Exemple 2

```
10 GR XPEN,YPEN
20 PLOT 0,0 TO XPEN-1,YPEN-1
```

Ce programme initialise ouvre une fenêtre graphique à la taille de l'écran et trace sa diagonale.

Action

Si *exp-num-1* et *exp-num-2* ne sont pas spécifiés, l'instruction GR initialise l'affichage graphique dans une fenêtre de 320 par 200 points. Sinon *exp-num-1* et *exp-num-2* sont la largeur et la hauteur de la fenêtre graphique à initialiser. *exp-num-1* doit être compris entre 320 et la largeur de l'écran. *exp-num-2* doit être compris entre 200 et la hauteur de l'écran.

Si une fenêtre graphique est déjà ouverte et que l'instruction GR est utilisée sans paramètre, cette fenêtre est seulement mise en avant plan. Même chose si les dimensions spécifiées sont égales à celles de la fenêtre graphique ouverte. Si une fenêtre graphique a été ouverte avec l'instruction HGR ou avec des dimensions différentes, celle-ci est préalablement fermée.

La couleur 15 est automatiquement sélectionnée lors de l'ouverture d'une fenêtre graphique.

Voir aussi : HGR, TEXT, XPEN, YPEN.

HGR — Instruction

Initialise la fenêtre graphique en 800x600 points.

Exemple

```
10 HGR
20 PLOT 10,10 TO 790,590
```

Action

L'instruction HGR initialise l'affichage graphique dans une fenêtre de 800 par 600 points. Si cette fenêtre a déjà été ouverte par l'instruction HGR, elle est seulement mise en avant plan. Si une fenêtre graphique a été ouverte avec l'instruction GR, celle-ci est préalablement fermée.

La couleur 15 est automatiquement sélectionnée lors de l'ouverture d'une fenêtre graphique.

Voir aussi : GR, TEXT.

MASK *exp-num-1, exp-num-2, exp-num-3* — Instruction

Permet de masquer une ou plusieurs composantes de base des couleurs.

exp-num-1	Masquage (1) ou non-masquage (0) de la composante rouge.
exp-num-2	Masquage (1) ou non-masquage (0) de la composante verte.
exp-num-3	Masquage (1) ou non-masquage (0) de la composante bleu.

Exemple

```
10 MASK 0,1,0
```

Après exécution de cette ligne, la composante verte de la couleur est masquée.

Action

L'instruction MASK n'a d'intérêt que seulement si les couleurs de 0 à 15 sont utilisées. Elle permet de masquer une ou plusieurs composantes de base de ces couleurs.

Chacune des trois expressions numériques *exp-num-1, exp-num-2* et *exp-num-3* est évaluée modulo 2 et peut prendre la valeur 1 ou 0 suivant que la couleur de base correspondante doit être masquée ou non.

Ce masque de couleurs n'est pas pris en compte par les instructions CLRG et FILL.

MOVE *exp-num-1, exp-num-2* — Instruction

Déplace les coordonnées du point courant dans la fenêtre graphique sans procéder à aucun tracé ni à aucun coloriage de points.

exp-num-1	Abscisse du point de destination.
exp-num-2	Ordonnée du point de destination.

Exemple

```
10 HGR
20 COLOR 2
30 CLRG
40 COLOR 1
50 ZERO% = 300
60 MOVE 0,ZERO%
70 FOR I% = 0 TO 500
80 X = I%*PI/500
90 PLOT TO I%, ZERO%+200*SIN(X**2)
100 NEXT I%
110 END
```

Ce programme permet de tracer par segments la courbe Y = SIN(X**2).

Action

L'instruction MOVE permet de déplacer le point courant au point de coordonnées (*exp-num-1, exp-num-2*) dans la fenêtre graphique, sans procéder à aucun tracé ou à aucun coloriage de points.

Les coordonnées du point peuvent être comprises entre -2147483648 et 2147483647 ; il s'agit d'un point virtuel éventuellement situé hors de la fenêtre graphique.

PEN [*exp-num*][,*var-chaîne*] Instruction

Sélectionne un point dans la fenêtre graphique.

exp-num	Option (entre 0 et 3).
var-chaîne	Variable chaîne pouvant recevoir la touche pressée.

Exemple 1

```
10 HGR
20 PEN
30 IF (XPEN>= 10 AND XPEN <=20) AND (YPEN>=10 AND YPEN <= 20 ) THEN 60
40 PRINT "Vous n'avez pas désigné le bon objet. Recommencez"
50 GOTO 20
60 PRINT "Réponse correcte"
```

La ligne 20 active le curseur de la souris et met le programme en attente de validation. Une fois que l'utilisateur clique sur l'emplacement choisi ou presse une touche, la ligne 30 est exécutée et vérifie que le point de coordonnées XPEN, YPEN se trouve bien à l'intérieur d'un carré de côté 11 dont le coin inférieur droit a pour coordonnées (10,10).

Exemple 2

```
10 HGR
20 COLOR 15
30 PEN 1
40 MOVE XPEN,YPEN
50 PEN
60 PLOT TO XPEN,YPEN
70 GOTO 40
```

A partir d'un premier point sélectionné, cet exemple trace le chemin de la souris.

Action

Si *exp-num* n'est pas spécifiée ou est égale à 0, l'instruction PEN attend la validation de la position du curseur de la souris. Une position du curseur dans la fenêtre graphique est validée en cliquant sur un bouton de la souris ou en appuyant sur une touche du clavier. La position est sauvegardée dans les variables système XPEN et YPEN.

Si exp-num est égale à 1 ou 2, la position courante du curseur de la souris est sauvegardée dans les variables système XPEN et YPEN. Il n'y a pas d'attente de validation.

Si *exp-num* est égale à 3, la souris n'est pas sollicitée. Ce sont alors les coordonnées du point courant qui sont sauvegardées dans les variables système XPEN et YPEN.

Lorsque *var-chaîne* est spécifiée, elle reçoit une chaîne d'un caractère à la fin de l'exécution de l'instruction PEN. Si la position du curseur a été validée en appuyant sur une touche du clavier, la chaîne contient le caractère correspondant à la touche. Sinon le code ASCII du caractère est le suivant :

ASC(*var-num*)	Etat boutons souris
0	Pas de bouton pressé
1	Bouton gauche pressé
2	Bouton droit pressé
4	Bouton du milieu pressé

Tableau 18 – Code validation position souris instruction PEN

Voir aussi : XPEN, YPEN.

PLOT *exp-num-1*, *exp-num-2* — Instruction

Affiche un point dans la fenêtre graphique.

exp-num-1	Abscisse du point.
exp-num-2	Ordonnée du point.

Exemple

```
10 HGR
20 COLOR 15
30 PLOT 10,20
```

Le point de coordonnées 10, 20 est affiché dans la couleur logique courante 15.

Action

L'instruction PLOT permet d'afficher le point de coordonnées (*exp-num-1*,*exp-num-2*) dans le fenêtre graphique.

Les coordonnées du point peuvent être comprises entre -2147483648 et 2147483647 ; il s'agit d'un point virtuel éventuellement situé hors de 1a fenêtre graphique.

Voir aussi : PLOT TO.

PLOT *exp-num-1*, *exp-num-2* TO *exp-num-3*, *exp-num-4* — Instruction

Trace un segment de droite dans la fenêtre graphique.

exp-num-1	Abscisse de l'origine.
exp-num-2	Ordonnée de l'origine.
exp-num-3	Abscisse de la fin.
exp-num-4	Ordonnée de la fin.

Exemple

```
10 HGR
20 COLOR 4
30 PLOT 10,20 TO 200,300
40 PLOT TO 400,400
```

Ce programme trace, en rouge, un segment de droite du point (10,20) au point (200,300), puis un autre segment du point courant (200,300) au point (400,400).

Action

L'instruction PLOT TO permet d'afficher un segment de droite du point de coordonnées (*exp-num-1*, *exp-num-2*) au point (*exp-num-3*, *exp-num-4*) dans la fenêtre graphique. Si les coordonnées du point de départ sont omises, ce sont celles du point courant qui sont prises en compte.

Si les coordonnées prennent des valeurs supérieures à celles permises par les dimensions de la fenêtre graphique, l'instruction est exécutée jusqu'aux limites et aucune erreur n'est signalée.

Voir aussi : PLOT.

POINT(*exp-num-1*, *exp-num-2*) Fonction

Retourne le numéro de la couleur logique du point désigné.

exp-num-1	Abscisse du point.
exp-num-2	Ordonnée du point.

Exemple

```
10 HGR
20 COLOR 1
30 CLRG
40 COLOR 12
50 PLOT 10,10 TO 500,500
60 I% = POINT(100,100)
70 PRINT I%
```

Après exécution de ce programme, la variable I% contient le numéro de la couleur logique dans laquelle est affiché le point (100,100). Ce point étant situé sur le segment précédemment tracé entre les points (10,10) et (500, 500), I% aura pour valeur 12.

Action

La fonction POINT retourne la couleur logique du point de coordonnées (*exp-num-1*, *exp-num-2*). *exp-num-1* et *exp-num-2* peuvent prendre valeurs comprises entre -2147483648 et 2147483647, mais si le point désigné est situé en dehors de la fenêtre graphique, la valeur 0 est retournée.

SETCOLOR *exp-num-1*, *exp-num-2*, *exp-num-3*, *exp-num-4* Instruction

Modifie la palette de couleurs de la fenêtre graphique.

exp-num-1	Numéro de la couleur logique compris entre 0 et 256.
exp-num-2	Niveau composante rouge compris entre 0 et 256.
exp-num-3	Niveau composante verte compris entre 0 et 256.
exp-num-4	Niveau composante bleue compris entre 0 et 256.

Exemple

```
10 HGR
20 SETCOLOR 100,255,0,0
30 COLOR 100
40 PLOT 10,10 TO 790,590
```

Après avoir modifié la couleur 100 de la palette graphique, ce programme trace un segment de droite du point (10,10) au point (790,590) eu rouge.

Action

L'instruction SETCOLOR permet de redéfinir une couleur de la palette graphique.

- *exp-num-1* est la couleur logique correspondant à l'entrée de la palette graphique à modifier. Si celle-ci est négative ou supérieure à 255, une *erreur 74* est signalée.
- *exp-num-2*, *exp-num-3* et *exp-num-4* définissent les niveaux des composantes rouge, verte et bleue à ranger dans la palette graphique. Si celles-ci sont négatives ou supérieures à 255, une *erreur 74* est signalée.

L'instruction SETCOLOR agit sur la fenêtre graphique si l'instruction TEXT n'a pas été exécutée après l'une des instructions GR et HGR. Elle agit sur la fenêtre texte si, ni l'instruction GR, ni l'instruction HGR, n'ont été exécutées depuis le lancement de SBASIC, l'exécution d'une commande RUN, NEW ou CLEAR. La modification de la palette graphique s'applique immédiatement à tous les points affichés dans la fenêtre graphique avec la couleur logique spécifiée dans l'instruction SETCOLOR.

Voir aussi : ARC, DRAW, FILL, PLOT, SYMBOL, TEXT.

SYMBOL *exp-num-1*, *exp-num-2* , *exp-chaîne* [, *exp-num-3*, *exp-num-4*, *exp-num-5*] Instruction

Permet d'afficher des textes de taille variable dans la fenêtre graphique.

exp-num-1	Abscisse du point de départ.
exp-num-2	Ordonnée du point de départ.
exp-chaîne	Chaîne de caractères à afficher.
exp-num-3	Largeur des caractères.
exp-num-4	Hauteur des caractères.
exp-num-5	Attributs.

Exemple

```
10 HGR
20 COLOR 0
30 CLRG
40 COLOR 3
50 SYMBOL 100,100,"ESSAT 1"
60 SYMBOL 150,150,"ESSAI 2",4,7,3
```

Le message « ESSAI 1 » est affiché à partir du point (100,100), puis « ESSAI 2 » est affiché à partir du point (150,150).

Action

L'instruction SYMBOL permet d'afficher des textes de taille variable dans la fenêtre graphique.

- *exp-num-1* et *exp-num-2* sont les coordonnées du point de départ du tracé et détermines le coin inférieur gauche du premier caractère,
- *exp-chaîne* contient le texte à afficher,
- *exp-num-3* et *exp-num-4* représentent respectivement la largeur et la hauteur des caractères. Ces paramètres doivent être compris entre 1 et 16. Ce sont les tailles d'un point élémentaire définissant les caractères tracés dans une matrice 6x8,
- *exp-num-5* représente les attributs du texte. Sa valeur est comprise entre 0 et 3 :

exp-num-5	Attributs du texte
0	Tracé horizontal, caractères droits
1	Tracé horizontal, caractères penchés
2	Tracé vertical, caractères droits
3	Tracé vertical, caractères penchés

Tableau 19 – Choix du tracé dans l'instruction SYMBOL

TEXT Instruction

Positionne la fenêtre SBASIC en avant plan.

Exemple

```
10 HGR
20 COLOR 15
30 PLOT 10,10 TO 500,500
40 A$=INCH$(-1): IF ASC(A$)=0 THEN 40
50 TEXT
```

L'affichage de la fenêtre graphique est activé à la ligne 10 puis un segment est affiché en blanc à la ligne 30 ; dès qu'une touche est frappée ligne 40, la fenêtre SBASIC revient en avant plan.

Action

L'instruction TEXT provoque l'affichage en avant plan de la fenêtre SBASIC.

Voir aussi : HGR, GR.

WINDOW *exp-num-1*, *exp-num-2*, *exp-num-3*, *exp-num-4* [,*exp-num-5*] Instruction

Définit la sous-fenêtre rectangulaire de tracé dans la fenêtre graphique ou la fenêtre SBASIC.

exp-num-1	Abscisse du coin inférieur gauche.
exp-num-2	Ordonnée du coin inférieur gauche.
exp-num-3	Abscisse du point supérieur droit.
exp-num-4	Ordonnée du supérieur droit.
exp-num-5	Option texte ou graphique (0 ou 1).

Exemple

```
10 HGR
20 COLOR 15
25 WINDOW 0,0,150,175,1
30 PLOT 10,20 TO 100,150
40 PLOT TO 200,200
```

Après avoir défini une sous-fenêtre allant du point de coordonnées (0,0) jusqu'au point de coordonnées (150,175) de la fenêtre graphique, ce programme trace, dans la couleur logique 15, un segment de droite du point (10,20) au point (100,150), puis un autre segment du point courant (100,150) au point (200,200), sans que le tracé puisse sortir de la sous-fenêtre définie.

Action

L'instruction WINDOW permet de définir une sous-fenêtre rectangulaire de tracé graphique.

- (*exp-num-1*, *exp-num-2*) et (*exp-num-3*, *exp-num-4*) sont les coordonnées de deux points diagonaux dans la fenêtre graphique. La valeur de chaque expression doit être comprise entre -2147483648 et 2147483647. Si les valeurs dépassant les limites de la fenêtre graphique, à savoir 0-799 et 0-599 en mode HGR, ou 0-319 et 0-199 en mode GR, elles sont ramenées à ces limites.
- *exp-num-5* est optionnel et vaut :
  - 0 pour indiquer une redéfinition de la fenêtre texte,
  - 1 pour indiquer une redéfinition de la fenêtre graphique.

Cette option permet de travailler en mode multifenêtre, à la fois en mode graphique et en mode texte, en redéfinissant une fenêtre chaque fois qu'on l'utilise. Si cette option n'est pas précisée, l'instruction WINDOW agit sur la fenêtre graphique si l'instruction TEXT n'a pas été exécutée après l'une des instructions GR et HGR. Elle agit sur la fenêtre texte si, ni l'instruction GR, ni l'instruction HGR, n'ont été exécutées depuis le lancement de SBASIC, l'exécution d'une commande RUN, NEW ou CLEAR.

L'instruction WINDOW s'applique aux ordres PLOT, SYMBOL, DRAW, ARC, FILL et GPUT. Tout tracé qui sort de la fenêtre est ignoré et n'apparaît pas dans la fenêtre graphique.

Voir aussi : ARC, DRAW, FILL, GPUT, PLOT, SYMBOL, TEXT.

XPEN — Variable Système

Variable système contenant l'abscisse du point désigné par la souris dans la fenêtre graphique après l'exécution de l'instruction PEN.

Exemple

```
10 PEN
20 IF (XPEN>= 10 AND XPEN <=21) THEN 50
30 PRINT "Vous n'avez pas désigné le bon objet. Recommencez"
40 GOTO 10
50 PRINT "Réponse correcte"
```

La ligne 10 met le programme en attende de validation. Une fois la souris cliquée, la ligne 20 est exécutée. Elle vérifie que le point désigné par la souris se trouve bien à l'intérieur d'une bande horizontale.

Action

XPEN est une variable système (de même que YPEN, ERR, ERL, PI ou DATE$) donnant la position, dans la fenêtre graphique, de la souris sur l'axe horizontal après la dernière exécution de l'instruction PEN.

On utilise la variable XPEN dans des expressions comme toute autre variable numérique, mais elle ne peut pas figurer à gauche du signe égal « = » dans un ordre d'affectation (LET explicite ou implicite).

Voir aussi : PEN, YPEN.

YPEN — Variable Système

Variable système contenant l'ordonnée du point désigné par la souris dans la fenêtre graphique après l'exécution de l'instruction PEN.

Exemple

```
10 PEN
20 IF (YPEN>= 10 AND YPEN <=21) THEN 50
30 PRINT "Vous n'avez pas désigné le bon objet. Recommencez"
40 GOTO 10
50 PRINT "Réponse correcte"
```

La ligne 10 met le programme en attente de validation. Une fois la souris cliquée, la ligne 20 est exécutée. Elle vérifie que le point désigné par la souris se trouve bien à l'intérieur d'une bande verticale.

Action

YPEN est une variable système (de même que XPEN, ERR, ERL, PI ou DATE$) donnant la position, dans la fenêtre graphique, de la souris sur l'axe vertical après la dernière exécution de l'instruction PEN.

On utilise la variable YPEN dans des expressions comme toute autre variable numérique, mais elle ne peut pas figurer à gauche du signe égal « = » dans un ordre d'affectation (LET explicite ou implicite).

Voir aussi : PEN, XPEN.

Annexe 1 : Liste des commandes, instructions et fonctions

A1 Fonctions mathématiques

ABS	Valeur absolue
ATN	Arc tangente
COS	Cosinus
EXP	Exponentielle
INT	Partie entière
LOG	Logarithme népérien
PI	Variable système contenant la valeur du nombre pi
RND	Générateur de nombres aléatoires
SGN	Signe
SIN	Sinus
SQR	Racine carrée
TAN	Tangente

A2 Fonctions graphiques

ARC	Tracé d'un arc de cercle
CLRG	Effacement de la fenêtre graphique
COLOR	Définition de la couleur courante
DASH	Définition de la nature du tracé
DRAW	Dessin de petits motifs
FILL	Coloriage d'un domaine
GR	Initialisation du mode graphique basse résolution
HGR	Initialisation du mode graphique haute résolution
MASK	Masquage de l'une des couleurs fondamentales
MOVE	Modification des coordonnées du point courant
PEN	Validation de la position de la souris
PLOT	Coloriage d'un point dans la couleur courante.
PLOT TO	Tracé d'un segment dans la couleur courante

POINT	Donne la couleur logique d'un point
SYMBOL	Dessin de lettres et de chiffres
TEXT	Affichage en avant plan de la fenêtre SBASIC
WINDOW	Définition de la sous-fenêtre de travail
XPEN	Abscisse de la position de la souris
YPEN	Ordonnée de la position de la souris

A3 Fonctions de chaînes

ASC	Code ASCII d'un caractère
CHR$	Caractère de code ASCII donné
FIELD	Définition d'un champ définissant un tampon-mémoire
INSTR	Recherche d'une chaîne dans une chaîne donnée
LEFT$	Extraction de la partie gauche d'une chaîne
LEN	Longueur d'une chaîne
LSET	Ecriture dans une variable tampon
LTRIM$	Suppression des blancs de tête
MID$	Extraction d'une partie de chaîne
RIGHT$	Extraction de la partie droite d'une chaîne
RSET	Ecriture dans une variable tampon
RTRIM$	Suppression des blancs de queue
SET	Ecriture dans une variable tampon
STRING$	Construction d'une chaîne répétitive

A4 Fonctions de conversion

ASC	Code ASCII d'un caractère
CHR$	Caractère de code ASCII donné
CVT%$	Conversion d'un entier en une chaîne de 2 octets
CVT$%	Conversion d'une chaîne de 2 octets en un entier
CVTF$	Conversion d'un réel en une chaîne de 8 octets
CVT$F	Conversion d'une chaîne de 8 octets en un réel
HEX	Conversion d'une chaîne hexadécimale en la valeur correspondante
STR$	Chaîne permettant d'écrire un nombre
VAL	Valeur d'une chaîne numérique

A5 Déclaration et affectation de variables

CLEAR	Libère la place occupée par des tableaux ou des variables
DATA	Introduit une suite de données lues par READ
DIM	Dimensionnement de tableaux
FTELD	Déclaration d'une variable tampon
LABEL	Définition d'une étiquette
LET	Affectation d'une variable
LSET	Ecriture dans une variable tampon
LOCAL	Localisation de variables et de tableaux

PTR	Adresse mémoire d'une variable
READ	Lecture de variables définies dans un DATA
RESTORE	Modification du pointeur utilisé par READ
RSET	Ecriture dans une variable tampon
SET	Ecriture dans une variable tampon
SUB	Définition d'une SUBroutine
SWAP	Permutation de deux valeurs

A6 Manipulation et modification de programme

BLOAD	Chargement d'un programme précompilé
CHAIN	Chargement et exécution de programme
COMPILE	Sauvegarde d'un programme sous forme précompilée
DELETE	Suppression de lignes de programmes
EDIT	Édition d'une ligne de programme
KILL	Destruction d'un programme se trouvant sur disquette
LIST	Listing d'un programme dans la fenêtre SBASIC
LOAD	Chargement d'un programme sauvegardé sous forme texte
NEW	Efface le programme se trouvant en mémoire
PLIST	Listing d'un programme dans le tampon d'impression
REFSUB	Édition des SUBroutines et des LABELs utilisés
REM	Documentation d'un programme
RENAME	Modification du nom d'un fichier disque
RENUMBER	Renumérotation d'un programme
RUN	Exécution du programme contenu en mémoire
SAVE	Sauvegarde, sous forme texte, du programme contenu en mémoire
TASVAR	Suppression des références inutiles
TRIVAR	Édition de la liste des références

A7 Mise au point de programmes

CONT	Continue l'exécution d'un programme arrêté par STOP
END	Arrête définitivement l'exécution d'un programme
ERL	Numéro de ligne où s'est produite la dernière erreur
ERR	Numéro de la dernière erreur
STOP	Arrêt provisoire d'un programme
TROFF	Suppression du mode trace
TRON	Activation du mode trace

A8 Tests, branchements et boucles

CALL	Appel d'une SUBroutine
EXECUTE	Exécution de 1'instruction contenue dans une chaîne de caractères
FOR	Initialisation d'une boucle
GOSUB	Appel d'un sous-programme
GOTO	Branchement à un numéro de ligne ou une étiquette

IF	Test d'une condition
LABEL	Définition d'une étiquette
ON ERROR GOTO	Gestion des erreurs
ON GOSUB	Appel conditionnel multiple d'un sous-programme
ON GOTO	Branchement conditionnel multiple
RESUME	Réactivation de l'instruction ON ERROR GOTO
RETURN	Fin de sous-programme ou de SUBroutine

A9 Entrée et sortie fichier

CLOSE	Libère un canal
FIELD #	Définition de variables associées à un tampon-mémoire
GET #	Transfert dans un tampon-mémoire du contenu d'un enregistrement
INCH$	Lecture d'un caractère depuis un fichier séquentiel
INPUT #	Lecture de valeurs depuis un fichier séquentiel
INPUT LINE #	Lecture d'une chaîne depuis un fichier séquentiel
LEN	Définition de la longueur d'enregistrement pour un fichier à accès direct
OPEN	Ouverture d'un fichier à accès direct Ouverture d'un tableau virtuel
OPEN NEW	Ouverture en écriture d'un fichier séquentiel
OPEN OLD	Ouverture en lecture d'un fichier séquentiel
POS	Position dans un canal
PUT #	Ecriture du buffer dans un enregistrement
PRINT #	Ecriture dans un fichier séquentiel

A10 Entrée clavier

INCH$(0)	Attente d'un caractère frappé au clavier
INCH$(-1)	Test du clavier et lecture d'un caractère
INPUT	Entrée de valeurs depuis le clavier
INPUT #0	Entrée de valeurs depuis le clavier sans affichage d'un « ? »)
INPUT LINE	Entrée d'une chaîne depuis le clavier (terminée par Retour-Chariot)

A11 Sortie sur écran et imprimante

CLOSE 0	Aiguille à nouveau les sorties vers la fenêtre SBASIC
CLS	Efface la fenêtre SBASIC et positionne le curseur en haut à gauche
CURSOR	Positionnement du curseur d'affichage dans la fenêtre SBASIC
DIGITS	Modification du format courant d'édition des nombres
LPRINT	Édition simultanée dans la fenêtre SBASIC et dans le tampon d'impression
POS(0)	Colonne où se trouve le curseur
PRINT	Édition dans la fenêtre SBASIC
PRINT #0	Édition dans la fenêtre BASIC ou dans un fichier
PRINT USING	Édition avec format
SPC	Affichage de blancs dans une instruction PRINT

TAB	Tabulation dans une instruction PRINT

A12 Fonctions diverses

BRON	Activation de l'action des touches Ctrl-C
BROFF	Désactivation de l'action des touches Ctrl-C
DPEEK	Lecture d'un octet double en mémoire
DPOKE	Chargement d'un octet double de la mémoire
EXEC	Exécution d'une ligne de commande Windows
EXIT	Retour au moniteur ou au système
SYSTEM	Retour au système MS-DOS
FRE(0)	Capacité de mémoire disponible
PEEK	Lecture d'un octet en mémoire
POKE	Chargement d'un octet de la mémoire
PORT	Définition du périphérique de sortie
USR	Appel d'un sous-programme en langage machine

A13 Variables système

DATE$	Date du système
ERL	Numéro de ligne où s'est produite la dernière erreur
ERR	Numéro de la dernière erreur
PI	Valeur du nombre Pi
XPEN	Abscisse de la position de la souris ou du curseur
YPEN	Ordonnée de la position de la souris ou du causeur

Annexe 2 : Liste des codes d'erreurs

Numéro de l'erreur	Signification
1	-
2	Le fichier requis est déjà utilisé
3	Le fichier spécifié existe déjà
4	Le fichier spécifié n'a pas été trouvé
5	Erreur en ouverture de fichier
6	Création fichier impossible
7	Tout l'espace disque a été utilisé
8	Fin de fichier rencontrée en lecture
9	Erreur de lecture sur disque
10	Erreur d'écriture sur disque
11	Fichier trop grand
12	Fichier protégé
13	Bloc de contrôle de fichier illégal
14	-
15	Numéro d'unité disque illégal
16	Unité disque non prête
17	Le fichier est protégé - accès refusé
18	Format fichier compilé incompatible
19	-
20	Type de fichier à accès direct non défini
21	Spécification de fichier illégale
22	Erreur en fermeture de fichier
23	Débordement de la table d'allocation - disque trop fragmenté
24	Numéro d'enregistrement inexistant
25	-
26	-
27	-
28	-
29	Enregistrement verrouillé
30	Type de données non concordant

Numéro de l'erreur	Signification
31	Hors donnée dans une instruction READ
32	Mauvais argument dans une instruction ON
33	-
34	-
35	-
36	-
37	-
38	-
39	-
40	Erreur de canal
41	Fichier déjà ouvert
42	Mode d'ouverture du fichier erroné
43	Le fichier n'a pas été ouvert
44	Erreur d'état fichier
45	Taille trop grande dans FIELD
46	-
47	-
48	-
49	Répertoire non trouvé
50	Instruction non reconnue
51	Caractère illégal dans la ligne
52	Erreur de syntaxe
53	Fin de ligne illégale
54	Numéro de ligne illégal (<=0)
55	Parenthèses non balancées
56	Référence illégale à une fonction
57	-
58	THEN manquant dans une instruction IF
59	-
60	Ligne non trouvée
61	Return sans CALL ou sans GOSUB
62	Erreur FOR NEXT
63	Ne peut continuer
64	Source absent
65	Mauvais fichier - ne peut être chargé
66	RESUME hors traitement d'erreur
67	Mauvais arguments dans DIGITS
68	Argument valide
69	Format fichier image invalide
70	Erreur de type dans une instruction PRINT USING
71	Format illégal dans une instruction PRINT USING
72	Types de opérandes incompatibles

Numéro de l'erreur	Signification
73	Expression illégale
74	Argument <0 ou >255
75	Argument <=0
76	Type de variable illégal
77	Référence à un tableau hors dimension
78	Référence à un tableau non dimensionné
79	Mauvais argument dans une instruction SWAP
80	Débordement mémoire
81	Débordement tableau
82	Débordement mémoire programme
83	Chaîne trop longue
84	Trop d'arguments dans une instruction SUB ou LOCAL
85	Chaîne vide dans une instruction EXECUTE
86	Chaîne trop longue dans une instruction EXECUTE
87	-
88	Redimensionnement d'un tableau
89	Effacement d'une ligne référencée
90	Fonction USR non définie
91	-
92	Argument virtuel dans l'instruction
93	Instruction SUB sans CALL
94	Mauvaise longueur de chaîne spécifiée
95	Instruction LOCAL sans SUB
96	Libération par CLEAR d'un tableau virtuel non fermé
97	Position curseur impossible
98	Mauvais argument dans l'instruction CLEAR
99	Type erroné dans les arguments d'un CALL
100	Expression trop complexe
101	Débordement virgule flottante
102	Argument trop grand
103	Division par zéro
104	Réel trop grand pour une conversion en entier
105	Argument négatif ou nul pour une fonction LOG
106	Erreur de conversion en entier dans une instruction INPUT
107	Racine carrée imaginaire
108	Erreur de conversion (nombre trop grand)
109	Débordement dans une opération entière
110	Numéro de couleur erroné
111	Numéro de port impossible
112	-
113	-
114	Ligne vide dans EDIT

Numéro de l'erreur	Signification
115	No de ligne absent dans EDIT
116	-
117	-
118	-
119	-
120	-
121	-
122	-
123	-
124	-
125	-
126	-
127	-
128	Erreur non récupérable

Annexe 3 : Liste des mots réservés

ABS
AND
APPEND
ARC
AS
ASC
ATN
BLOAD
BROFF
BRON
CALL
CHAIN
CHD
CHR$
CLEAR
CLOSE
CLRG
CLS
COLOR
COMPILE
COS
CURSOR
CVT$%
CVT$F
CVT%$
CVTF$
DASH
DATA
DATE$
DELETE
DIGITS
DIM

DPEEK
DPOKE
DRAW
EDIT
ELSE
END
ERL
ERR
ERROR
EXEC
EXECUTE
EXIT
EXP
FIELD
FILL
FOR
FRE
GET
GGET
GOSUB
GOTO
GPUT
GR
HEX
HGR
IF
INCH$
INPUT
INSTR
INT
KILL
LABEL
LEFT$
LEN
LET
LINE
LIST
LOAD
LOCAL
LOCK
LOG
LPRINT
LSET
LTRIM$
LTRM$

MASK
MID$
MOD
MOVE
NEW
NEXT
NOT
OLD
ON
OPEN
OR
OVERLAY
PDL
PEEK
PEN
PI
PLAY
PLOT
POINT
POKE
PORT
POS
PRINT
PTR
PUT
READ
RECORD
REM
RENAME
RESET
RESTORE
RESUME
RETURN
RIGHT$
RND
RSET
RTRIM$
RTRM$
SET
SETBLINK
SETCOLOR
SGN
SIN
SPC
SQR

STEP
STOP
STR$
STRING$
SUB
SWAP
SYMBOL
TAB
TAN
TEXT
THEN
TIME
TO
TROFF
TRON
UNLOCK
USING
USR
VAL
WINDOW
XPEN
YPEN
ZOOM

Annexe 4 : SBASIC sous Windows

Sous Windows, SBASIC est proposé en 2 formats : fenêtre ou ligne de commande.

La fenêtre SBASIC se rapproche le plus de la version originale sous MS/DOS au niveau de l'édition de ligne. Elle est complétée par un menu et une aide en ligne. Le fichier exécutable correspondant est SB32.EXE.

En ligne de commande, SBASIC tire partie des fonctionnalités de la console Windows. Le fichier exécutable est SBASIC.EXE.

SB32.EXE et SBASIC.EXE partagent le même moteur. Seule l'interface texte diffère.

A1 Fenêtre SBASIC (SB32.EXE)

Afin de garder la compatibilité avec les applications SBASIC sous MS/DOS, la fenêtre SBASIC affiche un écran de 25 lignes de 80 caractères. La taille de l'écran SBASIC peut être modifiée par l'instruction TEXT. Lorsque la taille de la fenêtre SBASIC est réduite, des curseurs permettent de naviguer horizontalement et verticalement à l'intérieure de cet écran.

Nouveau sous Windows, le curseur vertical permet de naviguer à travers les lignes de texte précédemment affichées. L'équivalent de 10 écrans est mémorisé. La molette de la souris permet aussi de naviguer verticalement. Dès que le curseur d'affichage se déplace, la fenêtre SBASIC est automatiquement recentrée autour de celui-ci.

A1.1 Edition ligne

Comme pour SBASIC sous MS/DOS, la combinaison de touches Ctr-R, utilisée au tout début d'une édition de ligne, permet de récupérer la dernière ligne entrée.

Nouveau sous Windows, les touches ↑ (flèche en haut) et ↓ (flèche en bas), utilisées au tout début d'une édition de ligne, permettent de retrouver une ligne parmi les 100 dernières lignes entrées. Si ces lignes sont utilisées sans modification, une séquence de ligne peut être simplement ré exécutée.

Nouveau aussi sous Windows, la combinaison de touches Ctr-V réalise un collé du texte présent dans le presse-papier.

A1.2 Menu fenêtre SBASIC

Les commandes SBASIC de sauvegarde et de chargement des programmes sont accessibles directement à partir du menu « Fichier » de la fenêtre SBASIC. Dans le menu « Programme » se trouvent les commandes contrôlant l'exécution.

A1.3 Aide

Un clic de souris sur un mot clé dans la fenêtre SBASIC ouvre une fenêtre d'aide sur celui-ci.

A2 Ligne de commande (SBASIC.EXE)

L'écran SBASIC prend la dimension de la fenêtre de commande Windows. Sa taille peut être modifiée par

l'instruction TEXT.

L'édition de ligne sous SBASIC en mode de ligne de commande est celle de la console Windows. Elle profite de l'historique de commandes et des fonctions de copié-collé.

A3 Photographie fenêtre graphique

Lorsque la fenêtre graphique est sélectionnée, la combinaison de touches Ctr-P effectue une photographie de cette fenêtre. Une fenêtre de dialogue permet de sauvegarder la photographie dans un fichier BMP.

A4 Appel fonctions DLL

L'instruction OPEN LIBRARY permet de charger une librairie à chargement dynamique (DLL). Les fonctions de la librairie peuvent être appelées par l'instruction CALL.

A5 Installation

Sous Windows, l'interpréteur SBASIC se présente sous la forme de 2 fichiers exécutables SB32.EXE (mode fenêtre) et SBASIC.EXE (mode ligne de commande). Son installation consiste simplement en la copie d'un de ces fichiers dans un répertoire de l'ordinateur.

Les différentes méthodes de lancement d'un programme au format .EXE peuvent être utilisées. Par exemples :

- Création, sur le bureau, d'un raccourci vers le fichier SB32.EXE ou SBASIC.EXE. Dans les propriétés du raccourci on spécifiera comme répertoire de travail, le répertoire contenant les fichiers SBASIC avec l'extension .BAS ou .BAC. L'interpréteur SBASIC se lance alors par un double-clic sur l'icône SBASIC :

  

  Un programme SBASIC peut être lancé directement en glissant le nom du fichier contenant le programme SBASIC sur l'icône SBASIC.

- Définition de SB32.EXE ou SBASIC.EXE comme programme associé aux extensions .BAS et .BAC. Un programme SBASIC se lance alors en double-cliquant, dans l'explorateur de fichier, sur le nom du fichier contenant ce programme.
- Pour que la commande SBASIC soit reconnue en ligne de commande, le fichier SBASIC.EXE doit être placé dans un répertoire spécifié dans le PATH (par exemple le répertoire C:\Windows).