SMT-GOUPIL

SBASIC
version 2.3

Imprimé en France

Les caractéristiques de ce produit sont indicatives et susceptibles de modification sans préavis.

SOMMAIRE

NOTE :

UNE ANNEXE (pages 151-155) PRECISE LES DIFFERENCES ENTRE LA VERSION DU SBASIC QUI FONCTIONNE SOUS LE SYSTEME D'EXPLOITATION FLEX ET CELLE QUI FONCTIONNE SOUS LES SYSTEMES D'EXPLOITATION MS-DOS et CP/M 86.

1. INTRODUCTION

Le présent SBASIC est très rapide et très complet. Il s'utilise comme la plupart des BASIC interprétés : les lignes, entrées au clavier, permettent de construire le programme qui sera par la suite lancé au moyen de la commande RUN. SBASIC a toutes les caractéristiques d'un BASIC interactif dont l'exécution en mode direct lui permet d'être utilisé comme calculateur.

Toutes les lignes entrées dans un programme SBASIC doivent commencer par un numéro de ligne. Les lignes sont automatiquement placées en ordre séquentiel, ce qui en permet une édition simple. Les lignes existant déjà dans un programme peuvent être supprimées en tapant simplement leur numéro suivi d'un retour chariot.

Nous recommandons à l'utilisateur de lire l'ensemble du présent manuel avant de travailler sous SBASIC. Nous avons supposé que le lecteur était déjà familiarisé avec la programmation BASIC. C'est la raison pour laquelle les exemples de ce manuel sont denses.

Cette version de SBASIC fonctionne sur le micro-ordinateur GOUPIL avec le système d'exploitation FLEX-9. Il est compatible avec le XBASIC du GOUPIL 2, tous les programmes écrits en XBASIC étant exécutables sous SBASIC, à condition que les fichiers correspondants aient l'extension .BAS (BAsic Source) et qu'ils ne contiennent pas l'instruction DEF FN.

SBASIC diffère du XBASIC par l'adjonction d'un certain nombre de mots clés et d'utilitaires 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.

- COMMENT CHARGER SBASIC ? -

Comme nous l'avons vu précédemment, SBASIC fonctionne sous le système d'exploitation FLEX-9. (*)

Il faut commencer par charger FLEX-9 de la manière suivante :

. mettre GOUPIL sous tension, le signe (+) apparaît à l'écran ;

. mettre la disquette FLEX-9 dans le lecteur 0 (lecteur du haut pour les lecteurs 5", lecteur de gauche pour les lecteurs 8"). Pressez la touche chargement < o— > .

. FLEX-9 affiche la date du jour. Au bout de quelques secondes, "+++" apparaît à l'écran.

. Frappez alors :

SBASIC

suivi d'un retour-chariot.

SBASIC met quelques secondes à se charger et affiche :

PRET

SBASIC est alors prêt à recevoir vos commandes et instructions.

(*) L'utilisateur disposant du SBASIC sous un autre système d'exploitation comme MS/DOS ou CP/M86 se réfèrera à l'annexe qui leur est consacrée en fin de Manuel.

2. DEFINITIONS

Plusieurs termes fréquemment utilisés par la suite, sont définis ci-dessous :

COMMANDE

Une commande est un ordre permettant d'exécuter immédiatement une opération spécifique. Habituellement la commande est lancée après que SBASIC ait affiché l'information "PRET". Alors l'ordinateur est prêt et attend que l'utilisateur lui indique quoi faire. L'utilisateur peut alors, soit entrer une commande, soit introduire une ligne de programme SBASIC

LIGNE

En SBASIC, chaque ligne de programme commence par un numéro et finit par un retour-chariot. Une ligne peut contenir une seule instruction ou plusieurs instructions séparées par le signe "deux points" (:).

AFFICHER

SBASIC affiche les informations à l'écran.

INSTRUCTION

Une instruction est un ordre que l'on fournit à l'interpréteur BASIC. Une instruction est habituellement exécutée par SBASIC lorsqu'elle est rencontrée dans un programme en cours d'exécution, alors qu'une commande est exécutée indépendamment de tout programme. Plusieurs instructions peuvent être utilisées comme des commandes, car elles peuvent être écrites soit dans un programme SBASIC, soit directement après le signal "PRET".

SIGNE

Signé : se dit d'un nombre qui possède un signe algébrique, positif ou négatif.

TAPER

L'utilisateur tape sur le clavier. (on dit aussi ENTRE au clavier).

CTRL C

Ce caractère est obtenu en maintenant la touche CTRL pressée pendant que l'on appuie sur la touche C. Cette instruction arrête le déroulement d'un programme sous SBASIC. Dans ce cas, l'instruction en cours d'exécution sera achevée et SBASIC se mettra en attente de la commande suivante. Au chargement cette instruction est inhibée, elle est validée par BREAK ON.

CTRL H

CTRL H s'utilise pour revenir sur le dernier caractère tapé. On peut alors remplacer le caractère pointé par un autre. Le même résultat est obtenu en frappant la touche "FLECHE VERS LA GAUCHE" du clavier GOUPIL.

CTRL X

CTRL X arrête la saisie de la ligne en cours de frappe (à partir de la position du curseur), et valide ce qui a été tapé depuis le début de la ligne, puis provoque un retour au début de la ligne suivante.

CTRL M

Est équivalent à un retour-chariot.

S'utilise pour effacer le caractère situé juste à gauche du curseur (c'est-à-dire le dernier caractère frappé).

3. CONVENTIONS

Plusieurs conventions seront utilisées ci-après pour rendre claire la définition des instructions et des commandes de SBASIC.

L'instruction que l'on définit dans le texte sera imprimée en lettres capitales. Les parenthèses angulaires "<" et ">" seront utilisées pour encadrer la partie obligatoire de l'instruction. Les parenthèses usuelles encadreront les parties optionnelles. On aura donc le schéma suivant :

<Eléments obligatoires> (Eléments optionnels)

Certaines lignes de SBASIC sont pour des raisons de mise en page écrites sur plusieurs lignes de texte : on entrera au clavier ces lignes en une seule fois, sans frapper de retour-chariot autre que celui terminant la dernière ligne.

4. NOTIONS FONDAMENTALES DE BASIC

4.1. LIGNES

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 32767 et chacune d'entre elles a un numéro unique. Lorsqu'un programme est exécuté en SBASIC, il commence à la ligne de plus petit numéro et se termine à la ligne de numéro le plus élevé.

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 en insérer éventuellement 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. En général il est indispensable de mettre un espace derrière les noms des variables de type "virgule flottante" et derrière les noms des routines sans argument (voir plus loin la définition de ces mots), dans tous les autres cas les espaces sont facultatifs mais rendent la lecture plus aisée.

Exemple de rédaction sans, puis avec espaces :

```
30 A=2*(B+4-7.2)/3.14+Y
40 A = 2 * (B+4-7.2) / 3.14 + Y
```

Exemple de lignes où un espace est obligatoire :

```
100 IF I%=0 THEN PRINT VITESSE ELSE PRINT TEMPS
```

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

Exemple comportant plusieurs instructions par ligne :

```
120 INPUT "VITESSE,TEMPS : " ; VIT,TPS : DIST=VIT*TPS
130 PRINT "DISTANCE ="; DIST
```

4.2. CONSTANTES

4.2.1. Constantes en virgule flottante

Toutes les valeurs en virgule flottante sont représentées de manière interne par une mantisse signée de 7 octets (55 bits et 1 bit de signe) et par un exposant signé d'un seul octet (7 bits et 1 bit de signe) permettant d'avoir des nombres compris entre 2**-127 et 2**128

Cela est approximativement équivalent à une mantisse de 17 chiffres décimaux avec un nombre compris entre 10**-37 et 10**38.

Souvent la forme la plus pratique pour représenter des nombres en virgule flottante est la notation scientifique. Le nombre est alors entré sous la forme d'une mantisse signée suivie de "E" ou "e", puis de l'exposant signé. SBASIC convertit automatiquement pour l'affichage sauf spécification contraire (voir l'instruction DIGITS) tout nombre en notation scientifique, s'il est plus grand ou égal à 1E+6 ou s'il est plus petit que 1E-2.

Quelques exemples de nombres en virgule flottante :

98.345 3.14159265 8975142 -1.23E+6 1E-04 1.0

Pour qu'une constante soit représentée de manière interne en virgule flottante, il faut qu'elle contienne un point décimal, ou qu'elle soit en notation scientifique, ou encore qu'elle soit trop grande pour être représentée en format entier. Si aucune de ces conditions n'est remplie, la constante sera convertie en entier (moindre capacité de stockage et plus grande rapidité d'exécution).

4.2.2. Constantes entières

Tous les entiers sont représentés de manière interne sous forme d'un nombre stocké sur 16 bits dont la valeur décimale est comprise entre -32768 et 32767.

Quelques exemples :

100 -12 32000 1 0

4.2.3. Constantes chaîne de caractères

Un autre type de constante est la chaîne de caractères. Elle est différente des autres constantes à la fois par la manière dont elle est définie et par son utilisation. Les constantes de chaîne de caractères apparaîtront le plus souvent dans les instructions PRINT et INPUT. Les constantes de chaîne sont définies en plaçant chaque caractère ASCII ou groupe de caractères (une chaîne) entre apostrophes ou guillemets. Une chaîne peut posséder entre 0 et 32767 caractères.

Quelques exemples :

```
"CHAINE"
"QUEL EST TON NOM ?"
"L'ECHO" (chaîne incluant une apostrophe)
'"VRAI"' (chaîne incluant des guillemets)
"H"      (chaîne formée d'un seul caractère)
""       (chaîne vide)
'"VRAI?"'
```

4.3. VARIABLES

Une variable est un identificateur de donnée pouvant prendre différentes valeurs. Le programmeur peut par exemple lui assigner une valeur puis la changer plus tard par exécution d'une instruction d'affectation. Les trois types de variables sont :

- Les variables en virgule flottante,

- Les variables entières,

- Les variables-chaînes de caractères.

Un nom de variable en virgule flottante consiste en une suite quelconque de symboles alphanumériques commençant obligatoirement par une lettre. Le blanc souligné (_) est également autorisé. N'utilisez donc pas de caractères accentués.

Exemples de noms de variables en virgule flottante :

K, INDEX1, DEP1, DEPART, NABUCHODONOSOR, JEAN_FRANCOIS.

Le second type de variable est la variable entière. Elle est définie en ajoutant au nom d'une variable, obéissant aux mêmes règles de définition que les variables en virgule flottante, le caractère %.
Les cinq noms de variables suivants sont des variables entières :

A%, KILO%, G95%, ER1% et NBRE%

Le troisième type de variable est la variable-chaîne de caractères (ou plus simplement chaîne). Il est défini en ajoutant au nom d'une variable-virgule flottante le caractère $.

Les cinq noms de variables suivants sont des variables chaîne.

A$, LIBELLE$, NOM$, FICHIER1$ et XX$

Chacun de ces types de variables peut être dimensionné comme décrit dans la section suivante. Le même nom de variable peut être utilisé dans un programme pour représenter une variable en virgule flottante, une variable de chaîne ou une variable entière. Par exemple "A" et "A$" définies ci-dessus, peuvent être utilisées simultanément dans le même programme et correspondront, bien entendu, à des variables différentes.

Remarquez que les caractères minuscules frappés sont ensuite automatiquement transformés en majuscules.

ATTENTION !!! Une restriction importante dans la définition des noms de variables :

Un nom de variable ne peut commencer par un mot réservé du SBASIC. Consultez la liste des mots réservés qui se trouve à la fin de ce document.

Exemples de variables erronées :

TABLEAU	commence par TAB
ERR	est un mot réservé
ERREUR%	commence par ERR
PRINT	est un mot réservé
PION$	commence par PI (3,14...)

4.4. DIMENSIONNEMENT

On peut aussi créer des tableaux de variables grâce à l'instruction DIM. Un tableau est un ensemble de variables de même type, flottant, entier ou chaîne.

L'instruction DIM a la syntaxe suivante :

<numligne> DIM variable (n1<,m1>) <,variable(n2<,m2>)>.....

où :
- variable est le nom d'une variable que l'on veut dimensionner,

- n1, m1, n2, m2, etc. peuvent être :

- une constante entière,
- une constante en virgule flottante, qui sera tronquée à l'entier inférieur,
- une variable entière,
- une variable en virgule flottante dont la valeur sera tronquée à l'entier inférieur,
- une expression dont la valeur sera tronquée à l'entier inférieur si elle n'est pas entière.

Exemple :

```
10 DIM AZERTY$(4), PRIX%(2,3)
```

crée :

. Un tableau de variables chaîne, à un seul indice (ou à une seule dimension) contenant cinq variables chaîne :

```
AZERTY$(0), AZERTY$(1), AZERTY$(2), AZERTY$(3), AZERTY$(4),
```

. Un tableau de variables entières à deux indices (ou à deux dimensions) :

```
PRIX%(0,0)  PRIX%(0,1)  PRIX%(0,2)  PRIX%(0,3)
PRIX%(1,0)  PRIX%(1,1)  PRIX%(1,2)  PRIX%(1,3)
PRIX%(2,0)  PRIX%(2,1)  PRIX%(2,2)  PRIX%(2,3)
```

```
10 DIM QUANT%(A*B+2)
```

évalue l'expression A*B+2 avec les valeurs courantes de A et B, tronque la valeur obtenue puis crée le tableau QUANT%. Si A vaut 3.3 et B vaut 1.7, on disposera de 8 variables QUANT%, de QUANT% (0) à QUANT% (7).

Les variables ainsi définies ne peuvent être utilisées avant que l'instruction DIM les définissant ne soit exécutée.

Tout tableau commence par l'élément 0.

Le même nom de variable peut être utilisé pour définir un tableau dimensionné et une variable non dimensionnée : SBASIC les considérera comme différents. Par contre, les tableaux à une et à deux dimensions ne peuvent partager le même nom de variable.

4.5. ASSIGNATION

On assigne des valeurs aux variables par les instructions LET, INPUT, ou par les combinaisons d'instructions READ et DATA. Ces instructions seront explicitées plus avant aux sections 6.1 et 6.4, mais nécessitent, dès à présent, d'être définies.

- La plupart des variables prennent des valeurs numériques ou des chaînes de caractères en utilisant l'instruction LET. Par exemple, l'instruction :

```
10 LET COMPTEUR=2
```

assigne une valeur en virgule flottante de "2" à COMPTEUR même si la constante "2" est représentée comme un entier dans la machine.

- Une instruction INPUT comme celle qui suit :

```
240 INPUT P1
```

affiche un point d'interrogation et met la machine en position d'attente d'une réponse d'un nombre flottant de la part de l'utilisateur. La donnée numérique tapée au clavier sera assignée à la variable P1.

- Les instructions READ et DATA doivent être utilisées ensemble. Brièvement, une instruction READ telle que :

```
100 READ TAUX
```

assignera à TAUX une nouvelle valeur chaque fois qu'elle sera rencontrée dans le programme. La première exécution de cette instruction assignera à TAUX le premier élément de données de la première instruction DATA du programme. La seconde exécution de l'instruction READ assignera le second élément de données de la première instruction DATA et ainsi de suite. Après que toutes les données aient été lues dans la première instruction DATA, la lecture se poursuit avec la seconde puis avec toutes celles apparaissant dans le programme. Si un READ est effectué après que le dernier élément de données du dernier ordre DATA ait été lu, l'erreur numéro 31 est signalée. Une instruction DATA semblable à la suivante devrait alors être ajoutée dans le programme :

```
500 DATA  1,18.6,3.14,5.80
```

On peut placer des chaînes alphanumériques dans une ligne de DATA, mais il est préférable de les placer entre guillemets.

```
600 DATA "JOUR","LUNDI","MARDI","MERCREDI","JEUDI"
```

4.6. OPERATEURS

Il y a trois catégories d'opérateurs disponibles, celle avec laquelle nous sommes le plus familiarisés étant celle des OPERATEURS ARITHMETIQUES (l'addition, la soustraction, la multiplication, la division et l'exponentiation).

La seconde classe est celle des OPERATEURS LOGIQUES. Ils sont utilisés pour réaliser bit à bit des opérations sur les quantités entières et par extension pour les tests conditionnels et les masques. Dans la mesure où ils opèrent 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.

Le troisième type d'opérateurs est celui des OPERATEURS DE RELATION qui sont également utilisés dans les tests conditionnels. Ils peuvent être utilisés par exemple dans une instruction IF pour déterminer quelle quantité est la plus grande.

Lorsque l'on veut comparer un nombre "en virgule flottante" et un entier, l'entier est d'abord converti en virgule flottante.

Chacune des trois classes d'opérateurs est décrite ci-après :

4.6.1. Opérateurs arithmétiques

SYMBOLE	EXEMPLE	SIGNIFICATION
+	X+Y	Ajoute X et Y
-	X-Y	Soustrait Y de X
*	X*Y	Multiplie X par Y
/	X/Y	Divise X par Y
**	X**Y	Elève X à la puissance Y
**	X**Y%	Elève X à la puissance Y%

Lorsqu'une expression mathématique contient plusieurs des symboles précédents, l'ordre des priorités est le suivant :

1. Exponentiation
2. Signe Moins
3. Multiplication et division
4. Addition et soustraction

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 "unaire" '(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 le plus à gauche qui est d'abord exécuté car SBASIC évalue les expressions de gauche à droite.

Exemple : RESULTAT = 6 + 2 * 4 donne à RESULTAT la valeur 14.

L'ordre peut toutefois être modifié par l'utilisation de parenthèses. SBASIC évalue d'abord les quantités entre les parenthèses les plus intérieures. Les parenthèses doivent être utilisées à chaque fois qu'il y a un doute au sujet de l'évaluation de l'expression.
Exemple : RESULTAT = (6 + 2) * 4 donne la valeur 32.

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.

Lorsque deux valeurs de type différent sont concernées, un entier et un flottant par exemple, l'entier est d'abord converti en flottant et l'opération est effectuée. L'opérateur d'exponentiation fait exception à cette règle et deux cas peuvent alors se produire. Tout d'abord, si la "base" (base puissance) est un entier, elle est convertie en virgule flottante. Notons bien que l'opérateur "**" fournit toujours un résultat en virgule flottante. Si la puissance est un entier, un algorithme rapide est alors appelé. Pour l'essentiel, il répète la multiplication.

Par contre si la puissance est en virgule flottante, l'algorithme suivant est utilisé :

X**Y = EXP (Y * log (X)).

Dans ce dernier cas, X ne peut être négatif.

4.6.2. Opérateurs logiques

Lorsque les opérateurs logiques sont utilisés sur un ou deux nombres, ils réalisent l'opération désirée sur les bits correspondants du ou des nombres. Si, par exemple, A% et B% sont égaux aux quantités binaires suivantes :

A%=(1100101111110110)
B%=(0111010111100100)

alors :

A% AND B%=(0100000111100100)
A% OR B%=(1111111111110110)
NOT A%=(0011010000001001)

Ces opérateurs peuvent être considérés comme des opérations bit à bit. Lorsqu'ils sont utilisés sous cette forme, ils opèrent sur un ou deux nombres pour donner un résultat numérique unique.

Les opérateurs logiques ont un effet totalement différent lorsqu'ils sont utilisés dans une expression qui est une condition de test d'une instruction IF-THEN. Dans ce cas, l'expression est évaluée logiquement (et non arithmétiquement) pour calculer si elle est vraie ou fausse. Une expression vraie a une valeur non nulle (=-1) et une expression évaluée fausse a la valeur 0.

Un ensemble d'instructions telles que :

```
22 IF A%>0 AND A%<10 THEN 40
30 PRINT "Condition non vérifiée" : GOTO 50
40 PRINT "A% est compris entre 0 et 10"
50 END
```

le programme se branchera à l'instruction 40 si et seulement si A% est compris entre 0 et 10. L'opérateur logique "AND" demande que la condition "A%>0" soit vraie, et que la condition "A%<10" le soit également.

Ci-dessous une liste des opérateurs disponibles :

NOT: exemple NOT X

Lorsqu'on opère sur des entiers, cet opérateur transforme chaque bit en son complément (les 1 sont remplacés par des 0 et les 0 par des 1). Ex : N%=NOT(0) rend N%=-1

AND: exemple X AND Y

Le résultat de cette opération est d'assigner à chaque bit du résultat, un "1" si chacun des bits de même rang vaut "1" dans X et Y. Lorsque AND est utilisé en test conditionnel, il faut que X et Y soient tous deux vrais pour que le résultat logique par AND soit vrai aussi.

OR : exemple A OR B

L'opération met un "1" dans chaque bit du résultat si l'un ou l'autre des bits de même rang dans A ou B en contient un. Utilisé dans un test conditionnel, le résultat sera vrai si A ou B le sont.

4.6.3. Opérateurs de relation

Comme son nom l'indique, ce groupe d'opérateurs teste la relation de variables à d'autres variables ou constantes. Les six symboles de relation reconnus par SBASIC sont :

SYMBOLE	EXEMPLE	SIGNIFICATION
=	X=Y	X est égal à Y
< >	X<>Y	X est différent de Y
<	X<Y	X est plus petit que Y
>	X>Y	X est plus grand que Y
<=	X<=Y	X est plus petit ou égal à Y
>=	X>=Y	X est plus grand ou égal à Y

Ces opérateurs sont souvent combinés avec les opérateurs logiques pour donner des tests complexes. L'instruction :

```
660 IF  A%=0  OR ((C-D)< 0.001) THEN 100
```

provoquera un branchement à l'instruction 100 si A% est égal à zéro ou si (C diminué de D est inférieur à 0,001).

4.6.4. Opérateurs de chaîne

Les opérateurs de chaîne rassemblent l'opérateur de concaténation (+), et les opérateurs de relation. L'opérateur '+' concatène deux chaînes (en les joignant) pour former une nouvelle chaîne.

```
Exemple : RESULTAT$="MICRO"+"-"+"ORDINATEUR" donne à
          RESULTAT$  la valeur "MICRO-ORDINATEUR".
```

Les opérateurs de relation, lorsqu'ils s'appliquent à des opérandes de type chaîne, utilisent l'ordre alphabétique. Si une chaîne est moins grande qu' une autre, cela implique qu'elle apparaîtra avant l'autre si elles sont affichées dans l'ordre alphabétique ASCII. Dans toute comparaison de chaîne les blancs sont SIGNIFICATIFS. Une chaîne de longueur zéro (chaîne nulle) est considérée comme étant plus petite que toute chaîne de longueur non nulle.

Tous les opérateurs de relations arithmétiques peuvent être utilisés avec les chaînes.

4.6.5. Priorités entre opérateurs

Les priorités entre opérateurs sont indiquées dans le tableau ci-dessous. L'opérateur en haut de liste a la priorité la plus élevée et la priorité décroît vers le bas. Les opérateurs de même priorité sont évalués par priorité, de gauche à droite.

1. ( ) Expressions placées entre parenthèses
2. ** Exponentiation
3. - Signe moins "unaire"
4. * / Multiplication et division
5. + - Addition et soustraction
6. < > = Opérateurs de relation
7. NOT L'opérateur NOT
8. AND L'opérateur AND
9. OR L'opérateur OR.

4.7. MODE COMMANDE ET MODE PROGRAMME

SBASIC peut fonctionner sous deux modes. Le premier mode utilisé le plus souvent utilise la commande RUN pour exécuter un programme déjà entré. On dit alors que l'on est en mode PROGRAMME.

L'autre mode est le mode immédiat. Dans ce mode, vous pouvez entrer une commande ou une instruction sans numéro de ligne et votre machine l'exécutera immédiatement : c'est le mode COMMANDE.

SBASIC distingue les deux modes par la présence ou l' absence du numéro de ligne.

Ainsi, par exemple, si vous tapez l'instruction :

```
100 PRINT "Dimanche"
```

il n'y aura pas d'exécution après le retour chariot parce que SBASIC suppose qu'il s'agit d'une partie d'un programme que vous êtes en train d'entrer et gardera la ligne en mémoire. Il ne l'exécutera que s'il reçoit la commande RUN.

Si, par ailleurs, vous tapez :

```
PRINT "Mardi"
```

L'ordre sera exécuté immédiatement parce qu'il n'y a pas de numéro d'instruction.

Les deux types d'instructions BASIC (commandes et instructions) croisent donc leurs chemins au niveau du mode immédiat. Les instructions peuvent être entrées sans numéro de ligne et utilisées comme si elles étaient des commandes. Certaines commandes peuvent être utilisées dans les lignes d'un programme. Ceci se fait après que SBASIC ait affiché "PRET", lorsqu'il attend vos instructions.

4.8. REMARQUES

Utiliser fréquemment des remarques dans les programmes est une bonne habitude de programmation. Cela facilite la compréhension de votre programme pour les futurs lecteurs, et pour les modifications ultérieures.

Les remarques peuvent être placées dans un programme en utilisant l'instruction REM. Lorsque le BASIC rencontre une instruction REM il l'ignore tout ce qui se trouve après elle jusqu'à la prochaine fin de ligne. Cette instruction peut se trouver seule sur une ligne de programme et son numéro peut être référencé ailleurs dans le programme (par une instruction GOTO par exemple), bien que cela ne soit pas conseillé .

Exemple : 520 REM * début du programme principal *

Il n'est pas autorisé de placer d'instructions "REM" sur une ligne de DATA.

5. COMMANDES

Les commandes ne sont pas utilisées dans les programmes SBASIC. Ce sont des instructions directement exécutables que l'on peut utiliser lorsque SBASIC est en mode COMMANDE, après affichage du signal "PRET".
Lorsqu'une commande est entrée dans votre machine, son exécution est immédiate.

5.1. LES COMMANDES PURES

Vous trouverez ci-après la liste des commandes disponibles existant en mode direct. Elle ne peuvent donc figurer dans les lignes d'un programme. Par contre, toutes les instructions de SBASIC peuvent s'exécuter en mode direct après le signal "PRET" et ont, dans ce cas, le même rôle qu'une commande.

BREAK ON ou BR ON :

Valide la possibilité d'arrêter un programme en cours de traitement par CTRL-C, et permet l'affichage des messages d'erreur disque (le fichier ERRORS.SYS doit, dans ce cas, être présent sur le disque système).

En cas d'arrêt, le programme réside toujours en mémoire, sa reprise est possible avec la commande CONT. Si l'arrêt survient sur un INPUT, la reprise s'effectue au début de la ligne où est situé l'instruction INPUT.

BREAK OFF ou BR OFF :

Invalide le CTRL-C. Cette option est prise par défaut au moment du chargement du SBASIC.

CONT

La commande CONT (CONTinue) est utilisée pour faire redémarrer un programme après qu'il ait été arrêté par un STOP ou un CTRL C. L'instruction STOP peut être placée n'importe où dans un programme, mais aussi, un CTRL C peut être tapé au moment où celui-ci attendait une introduction de données requises par l'instruction INPUT.
Après cette commande, l'exécution du programme ne peut pas reprendre s'il y a eu une erreur ou si l'on a entré ou modifié des lignes du programme après l'avoir arrêté. Si le programme est arrêté par un STOP, CONT le fait poursuivre à partir de l'instruction suivante. Si le programme a été arrêté sur une instruction INPUT par un CTRL C , CONT fera reprendre l'exécution au début de l'instruction INPUT. Sur un STOP ou sur un CTRL C au moment d'un INPUT, CONT fonctionne.

EDIT

Pour modifier une ligne de votre programme taper EDIT suivi du numéro de la ligne à modifier. La ligne indiquée s'affiche alors sur l'écran, avec le curseur positionné en début de ligne.

Il est alors possible :

- de déplacer le curseur à l'aide des touches FLECHE GAUCHE et FLECHE DROITE du clavier central.

- d'effacer le caractère pointé par le curseur, à l'aide de la touche (EFF)

- d'effacer le dernier caractère entré avec la touche (⌫) RUBOUT

- de remplacer le caractère pointé par le curseur en tapant le nouveau caractère

- de se mettre en mode insertion à l'aide de la touche (INS). Dès lors tout caractère tapé au clavier viendra s'insérer dans la ligne juste avant celui où se trouve le curseur.

Le fait de refrapper sur cette touche permet de sortir de ce mode.

L'utilisation de l'une des touches (FLECHE GAUCHE, FLECHE DROITE, EFF, RUBOUT)) ne fait pas sortir du mode insertion.

L'utilisation de la touche (RETOUR-CHARIOT) permet de sortir du mode EDIT en validant toute la ligne QUELQUE SOIT LA POSITION DU CURSEUR .

On est implicitement sous éditeur. Les seuls caractères acceptés sont les caractères applicables et les caractères de CTRL de l'éditeur. Les caractères accentués sont acceptés et sont maintenant complètement gérés.

TAB (CTRL I) Avance le curseur de 8 en 8 (en fin de ligne, on passe au début de la ligne courante).

CTRL R Ré-édition de la ligne courante ; en édition, on passe alternativement du début en fin de ligne et vice-versa.

CTRL C Abandon de la modification.

CTRL X Valide la partie de la ligne jusqu'au curseur.

< ↵ > Valide la ligne complètement (RETOUR-CHARIOT).

< → >
et
< ← > Déplacement du curseur sans effacement de caractères.

INS Bascule le mode Insertion/Remplacement.

EFF Efface le caractère sous le curseur (sans effet si fin de ligne).

< ⌫ > Efface le caractère à gauche du curseur (sans effet si en début de ligne) (RUBOUT)

Les instructions INPUT et INPUT LINE utilisent l'éditeur de ligne de la même façon.

EDITEUR : est isolé et indépendant (implanté en $E000 à $E3FF).

EXECUTE :

S'utilise sans restrictions. Accepte une ligne comme :
Exemple : 100 A$ = "200 A = A + 1:PRINT A"
120 EXECUTE A$

Ces 2 instructions génèrent une ligne de programme.
(voir l'instruction de réservation de place OVERLAY)

EXIT :

Cette commande est utilisée pour sortir de SBASIC et se placer sous moniteur GOUPIL.

FLEX-9 ou FLEX

La commande FLEX-9 (ou FLEX) est utilisée pour sortir du SBASIC et retourner au système FLEX-9

LIST

```
LIST       (liste tout le programme)
LIST 10    (liste la ligne 10)
LIST 50,80 (liste les lignes de 50 à 80)
```

Mais :

```
LIST ,60   (liste du début à la ligne 60)
LIST 100,  (liste de la ligne 100 jusqu'à la fin)
```

LIST est utilisée pour afficher des lignes du programme comme le montre les exemples. On peut afficher tout le programme ou seulement des parties de celui-ci. Lorsqu'un programme est en cours d'affichage on peut arrêter l'affichage en tapant sur la barre d'espacement, le faire reprendre en tapant à nouveau sur cette même barre ou l'interrompre définitivement en tapant CTRL M ou RETOUR CHARIOT.

PLIST

Identique à la commande LIST, mais avec édition sur l'imprimante. PLIST utilise le programme Moniteur de pilotage d'imprimante.

NEW

Lorsque la commande NEW est exécutée, le programme en mémoire est effacé. Après avoir exécuté cette commande la mémoire est prête à recevoir un nouveau programme.

RUN

La commande RUN ordonne à votre machine de commencer l'exécution du programme en mémoire. Lorsque vous lancez un programme avec cette commande, toutes les variables sont initialisées à zéro et le pointeur des instructions DATA est positionné sur la première donnée.

SAVE

SAVE est utilisé pour stocker, sous forme source (c'est-à-dire sous forme TEXTE) TOUT ou une PARTIE d'un programme BASIC, sur disque. Le nom du fichier doit être donné entre apostrophes ou guillemets et dans la forme standard au FLEX soit : UNITE.NOM.EXTENSION. L'unité par défaut est celle de travail et l'extension par défaut est .BAS. Tous les programmes SBASIC sauvés à l'aide de la commande SAVE peuvent être utilisés par n'importe quel programme FLEX-9 travaillant avec des fichiers texte standards.

NOTE IMPORTANTE : La commande SAVE effacera tout fichier de même nom existant déjà sur le disque et ceci sans prévenir !!!

La syntaxe générale de la commande SAVE est la suivante :

SAVE <"fichier"> (<numligne1>, <numligne2>, <numligne3>)

Elle permet de sauvegarder sous le nom de <"fichier"> les lignes dont le numéro est compris entre <numligne1> et <numligne2>, en translatant tous les numéros de ligne du programme de la valeur <numligne3>-1.

Sur disque, les lignes du programme conservées dans <"fichier"> seront donc numérotées entre <numligne1>-<numligne3>+1 et <numligne2>-<numligne3>+1. Il faut évidemment que <numligne3> soit inférieur ou égal à <numligne1> pour que tous les numéros de ligne de <fichier> soient positifs. (Si <numligne3> = <numligne1>, la première ligne du fichier programme conservé sur disque aura pour numéro 1)

Les valeurs par défaut sont :

<numligne1>=1
<numligne2>=32768
<numligne3>=1

Exemple:

SAVE "prog"

sauvegarde tout le programme dans le fichier "PROG.BAS" sur l'Unité de Travail.

SAVE "prog",,10.

sauvegarde dans ce même fichier et modifie la numérotation

SAVE "prog"10,1000

sauvegarde dans le même fichier les lignes comprises entre 10 et 1000 sans renumérotation.

5.2. LES UTILITAIRES SBASIC

La commande "+" permet à SBASIC d'envoyer le reste de la ligne de la commande à FLEX-9. Il s'agit d'une commande dangereuse car certains programmes utilitaires FLEX-9 se chargent dans la même zone mémoire que le SBASIC. Si un utilitaire FLEX-9 est exécuté à partir de SBASIC, il faut s'assurer que l'utilitaire se charge bien dans l'espace des commandes utilitaires du FLEX-9. On peut également lancer l'utilitaire de FLEX-9 à l'intérieur d'un programme BASIC à l'aide de l'instruction EXEC (voir plus loin).

Ne pas utiliser les commandes du Type NEWDISK, COPY, BACKUP qui détruisent l'espace mémoire occupé par le SBASIC !

Les utilitaires de SBASIC qui se présentent comme des commandes FLEX-9 ont été conçus pour vous apporter une aide efficace dans la phase de mise au point de votre programme. Ils travaillent sur le programme existant en mémoire. L'utilitaire COMPILE vous permettra en outre de protéger vos programmes.

Les utilitaires existant actuellement sont les suivants : TRIVAR, REFSUB, SBRENUMB et COMPILE. D'autres utilitaires viendront compléter cet ensemble.

TRIVAR

Appel sous SBASIC :

+TRIVAR

Cet utilitaire permet de sortir sur l'écran, ou sur l'imprimante si vous tapez "+P TRIVAR", la table des symboles du programme triée dans l'ordre alphabétique : noms de variables, noms de sous-programmes, noms de tableaux, labels existant dans le programme ; chaque nom étant suivi de la liste des numéros de ligne où il apparaît : ce sont les références croisées du programme. Un essai sur un programme vous montrera l'intérêt de cet utilitaire.

SBRENUMB

Appel sous SBASIC :

+SBRENUMB (<nl>,<pas>,<ld>,<lf>)

Cet utilitaire permet de renuméroter en mémoire, tout ou partie du programme. Il peut être appelé avec 4 arguments et effectue une renumérotation des lignes du programme en mémoire pourvu qu'elles ne se chevauchent pas. Les arguments ont la signification suivante :

<nl> : nouveau numéro de la première ligne à renuméroter
<pas> : pas de la renumérotation
<ld> : ancien numéro de la première ligne de la zone à renuméroter
<lf> : ancien numéro de la dernière ligne de la zone à renuméroter

Tous ces arguments ont une signification par défaut :

<nl>=10, <pas>=10, <ld>=1, <lf>=32767 (renumérotation de tout le programme de 10 en 10)

Ainsi par exemple :

+SBRENUMB ,1,100,200 est équivalent à +SBRENUMB 10,1,100,200
+SBRENUMB 1,5,,400 est équivalent à +SBRENUMB 1,5,1,400

La renumérotation d'un programme de 400 lignes est extrêmement rapide.

ATTENTION :

SBRENUMB ne reloge pas les lignes. Il est impossible de faire chevaucher les numérotations ancienne et nouvelle. Si cela était le cas, cette commande serait sans effet.Il n'y a pas de message de bonne fin. Mais il y a un message d'impossibilité de renumérotation (pour cause de chevauchement ou de syntaxe). Dans les programmes comportant ERL, SBRENUMB doit être employé avec précaution :
La ligne suivante sera renumérotée :

Exemple : 32000 IF ERL=5500 THEN RESUME 5520.

Dans cet exemple, même le No de ligne 5500 sera renuméroté !

Si les lignes de programmes se chevauchent, utilisez de préférence l'utilitaire "SUPRENUM" du package "BASUTIL".

COMPILE

La commande COMPILE est utilisée pour sauver un programme sur disquette sous une forme codée. Le nom du fichier doit être spécifié sous la forme du standard FLEX-9 (UNITE.NOM.EXTENSION). Le nom du fichier NE DOIT PAS être entouré de guillemets dans l'ordre compile, car c'est un utilitaire FLEX-9. L'unité par défaut sera celle de travail et l'extension par défaut 'BAC' (BAsic Compilé). Le programme résultant sera moins encombrant que le même programme sauvegardé par la commande SAVE si l'option "C" est utilisée. Ce programme compilé pourra être chargé, par exemple, par la commande BLOAD (voir plus loin) et exécuté grâce a la commande RUN.
COMPILE s'utilise avec la syntaxe suivante :

+COMPILE <nomfic> (<ld>, <lf>, <nld>) (+OPTIONS)

où
- <nomfic> est un nom de fichier.
- <ld>, <lf> sont les numéros de ligne début et fin de la zone à compiler (en fait à précompiler), par défaut respectivement 1 et 32767.

- <nld> (nouveau numéro de ligne début) est un paramètre de déplacement. Ce sera, dans le code compilé, le numéro de la première ligne du segment traité. Par défaut <nld> vaut 1.

- Les options possibles sont :

C : COMPILE supprime les REM et les blancs. Le programme reste listable (mais pas modifiable).

S : COMPILE supprime la table des symboles. Le programme devient alors illisible.

L'option S est incompatible avec les chargements partiels (OVERLAY).

P : rend le programme non listable. Il peut seulement être exécuté.

Notons que si aucune option n'est sélectionnée, vous retrouverez votre programme en mémoire dans son état d'origine après avoir chargé avec BLOAD, le chargement étant alors deux à trois fois plus rapide que par l'instruction LOAD.

5.3. PSEUDO-COMMANDES

Ces commandes habituellement utilisées en mode direct peuvent s'exécuter dans SBASIC à l'intérieur d'un programme comme toute instruction normale.

KILL

L'ordre KILL est utilisé pour supprimer un fichier disque. La syntaxe en est :

<numligne> KILL <expression chaîne>

où l'expression chaîne est le nom du fichier à supprimer. L'extension par défaut est .BAS et le disque par défaut est le disque de travail. Cette instruction peut aussi être utilisée en mode immédiat.

Exemples :

```
100 KILL "MENU"
120 PROG$="0.MENU2.DAT":KILL PROG$
```

La ligne 100 détruit le fichier intitulé MENU.BAS sur l'unité de travail puis le fichier MENU2.DAT sur l'unité 0.
Si le fichier spécifié n'existe pas, une erreur 4 est générée.

RENAME

L'ordre RENAME est utilisé pour renommer un fichier disque. Il peut être utilisé dans un programme ou en mode immédiat.

La syntaxe en est :

<numligne> RENAME <expression chaîne>, <expression chaîne>

où la première chaîne précise le nom du fichier à renommer et la seconde chaîne le nouveau nom qu'il doit porter. L'extension par défaut est l'extension .BAS, et le disque par défaut, le disque de travail.

Exemple :

```
225 RENAME "CLIENT", "ANCLIENT"
```

Cette ligne permet de renommer le fichier CLIENT.BAS en ANCLIENT.BAS. Si CLIENT.BAS n'existe pas, l'erreur 4 est signalée. Si ANCLIENT.BAS existe déjà, c'est une erreur 3.

EXEC

L'instruction EXEC permet d'exécuter les utilitaires FLEX-9 qui se chargent dans l'espace des commandes utilitaires ($C100). La syntaxe en est :

<numligne> EXEC, <"commande">

où "commande" a une syntaxe identique à celle de la commande FLEX-9, par exemple :

```
300 EXEC, "TTYSET PS=0"
320 EXEC, "ASN W=0"
```

L'exécution de la ligne 300 permet d'effectuer une pause d'affichage à chaque fin d'écran. La ligne 320 assigne en 0 le lecteur de Travail par défaut. La commande est transmise et exécutée comme si elle avait été tapée directement sous FLEX-9. Il faut noter que seules les commandes résidant dans l'espace des commandes utilitaires peuvent être utilisées, sinon le programme serait écrasé lors du chargement de la commande.

EXEC ne doit donc pas utiliser NEWDISK, COPY ou BACKUP qui se charge dans la zone du SBASIC.

TRON et TROFF

TRON active la fonction TRACE utilisée pour rechercher les erreurs dans les programmes. Elle affiche les numéros des lignes du programme exécuté. TROFF ou NEW désactivent TRACE.

Ces instructions sont aussi utilisables en mode programme avec la syntaxe :

<numligne1> TRON
<numligne2> TROFF

Exemple : 500 TRON:REM mise en oeuvre mode TRACAGE.

TRON se met à afficher les numéros des lignes exécutées à partir de la ligne 500 et s'arrêtera dès rencontre de l'ordre contraire "TROFF".

DELETE

Permet de supprimer un certain nombre de lignes du programme. La syntaxe en est :

DELETE <numligne1>,<numligne2>

supprime de la mémoire centrale les lignes dont le numéro est compris entre <numligne1> et <numligne2>. Par défaut, <numligne1> est la première ligne du programme et <numligne2> la dernière :

DELETE 12000,15000

supprime toutes les lignes comprise entre 12000 et 15000.

DELETE 4000,

supprime toutes les lignes depuis la ligne 4000 jusqu'à la fin.

DELETE s'utilise aussi en mode programme avec la syntaxe :

<numligne> DELETE <numligne1>, <numligne2>

Les conventions sont les mêmes que précédemment (il est nécessaire, dans ce cas, de fixer une taille réservée au programme avec l'instruction OVERLAY. Reportez-vous au paragraphe traitant l'OVERLAY).

ATTENTION : en mode programme, on ne peut pas effacer les lignes précédant celle dans laquelle l'ordre DELETE apparaît. Si cela était, une erreur 89 serait signalée par SBASIC.

LOAD

La commande LOAD est utilisée pour charger un fichier programme de type "texte" de la disquette vers la mémoire sans effacer celui qui s'y trouve déjà. Le nom du fichier doit être entré apostrophes ou guillemets et dans la forme du standard FLEX-9 (UNITE.NOM.EXTENSION). L'unité par défaut est celle de travail et l'extension par défaut est .BAS (BAsic Source).

Avec la syntaxe :

LOAD <"fichier">

SBASIC charge le programme sans effacer la mémoire, ce qui permet de fusionner plusieurs programmes (MERGE). Les lignes du programme vont alors se placer en fonction de leur numéro, entre les lignes du programme existant en mémoire. En cas de conflit de numéros, les lignes du programme <"fichier"> remplacent celles du programme que contenait la mémoire.

Mais cet ordre peut s'utiliser aussi avec la syntaxe :

LOAD <"fichier"> (<numlign1>, <numlign2>, <dep>)

ou <numlign1> et <numlign2> représentent les numéros des lignes entre lesquelles on autorise le chargement des lignes du programme.

Ce sont les numéros de ligne de l'origine sur disque.

"fichier", et <dep>, un nombre positif qui sera ajouté lors du chargement aux numéros de ligne se trouvant dans le programme (fichier) afin de les situer entre <numlign1> et <numlign2>. En cas de conflit avec des lignes du programme se trouvant en mémoire ce sont les lignes de <"fichier"> qui s'imposent. Les lignes du programme <"fichier"> qui après la renumérotation dûe à l'offset ne se trouvent pas dans l'intervalle <numlign1>, <numlign2>, ne seront pas prises en compte.

Les valeurs par défaut de ces quantités sont :

<numlign1>	1
<numlign2>	32768
<dep>	0

Enfin, cet ordre peut aussi s'utiliser dans un programme pour charger des sous-programmes fonctionnels ou utilitaires avec la syntaxe :

<numligne> LOAD <"fichier"> (<numlign1>,<numlign2>,<dep>)

avec les mêmes conventions. Se reporter au mode d'emploi de OVERLAY pour voir l'utilisation de cette instruction.

BLOAD

BLOAD est une instruction de SBASIC qui peut s'utiliser en mode programme avec la syntaxe suivante :

<numligne> BLOAD <"nomfic"> (,,<dep>)

ou en mode immédiat de la même façon mais sans <numligne> (",," assure une conformité syntaxique avec LOAD et SAVE).Cette instruction permet de charger en mémoire le fichier compilé "nomfic". Ce fichier doit avoir été compilé par l'utilitaire COMPILE du SBASIC .

- <"nomfic"> est un nom de fichier analogue à celui qui apparaît dans COMPILE, l'extension étant encore .BAC par défaut. (Ici le nom du fichier doit être spécifié entre parenthèses)

- <dep> est la quantité qu'il faut ajouter au numéro de la première ligne du programme que vous souhaitez charger pour le reloger dans une nouvelle zone de numérotation. On appelle ce paramètre offset ou déplacement.

ATTENTION : Lorsque BLOAD est utilisé en mode programme, il ne peut charger à un numéro de ligne inférieur à celui de la ligne où il apparaît. Si tel était le cas, une erreur 89 serait signalée par SBASIC.

6. INSTRUCTIONS

Toutes les instructions SBASIC listées ci-après sont regroupées par fonctions ou par similitudes. On trouvera à la droite de chaque instruction une description de son utilisation générale, description suivie d'un ou plusieurs exemples. Viennent ensuite la définition et les explications.

6.1. ASSIGNATION

DATA

<numligne> DATA <expression>(,<expression>,<expression>,...)
où <expression> est une expression numérique ou une chaîne .

```
50 DATA -3.556E-5, 0, 2, 4, 59E11
60 DATA AVRIL,200,"C'EST TOUT"
70 DATA " 100","1000","10,000"
```

L'instruction DATA fournit les données qui seront lues dans la suite du programme par une instruction READ. Les données sont lues de gauche à droite et commencent naturellement par le premier élément du premier DATA. Chaque fois qu'une instruction READ est rencontrée dans le programme, les éléments apparaissant dans les DATA sont assignés aux variables apparaissant dans l'ordre READ (Attention au respect des types de variables numériques et alpha). Les instructions READ commenceront par prendre les données de la première instruction DATA qui apparaît dans le programme. Lorsque tout le contenu de cette instruction aura été lu, READ saute à l'instruction DATA suivante, et ainsi de suite. Si par exemple nous lançons un programme contenant la ligne 50, la première fois qu'un READ sera exécuté, la valeur lue sera -3.556E-5. La valeur suivante sera 0 et ainsi de suite jusqu'à la fin du premier DATA qui sera suivi par le début du second DATA. Si une chaîne de caractères contenant des blancs ou des virgules est nécessaire, elle doit être placée entre apostrophes ou guillemets comme dans l'instruction 70. L'instruction DATA doit être la première et la seule instruction dans une ligne de programme. Elle n'est pas utilisable en mode immédiat.

LET

<numligne> (LET) <variable > = <expression>

```
10 LET CURX%=3.5
25 LET HAUTEUR%=27.2 * H1 / (5.4E7-LARGE)
70 LET DOMINIQUE$="SAMEDI ET DIMANCHE"
75 LET RANG(5,LIG%)=0    (variable dimensionnée)
80 ARRONDIPY=3.1416      (LET implicite)
90 Y%=Y%+1               (Incrémentation de variable)
95 Z=Y%                  (Conversion de type de variable)
96 Y%=INT(FRANC)         (Partie entière de l'expression)
```

L'instruction LET (facultative) assigne une valeur à une variable. Toute variable peut se faire assigner une valeur par l'utilisation de cette instruction. La valeur peut être une constante comme dans l'instruction 10, ou une expression complexe comme dans l'instruction 25. Notons que les instructions 80 à 96 ont omis le mot "LET" : c'est un LET implicite. C'est en fait une facilité de l'interpréteur SBASIC qui permet d'omettre l'écriture de LET. Notons également qu'en ligne 95 la variable assignée est d'une précision différente de la valeur reçue. Dans cet exemple particulier l'expression entière est convertie en virgule flottante après évaluation, et l'assignation est ensuite effectuée. En ligne 96, c'est l'inverse qui se produit. L'expression en virgule flottante est convertie en entier et l'assignation a ensuite lieu.

READ

<numligne> READ <variable> (,<variable>,<variable>,...)

```
50  DATA 5.5, 18.6, 33.333
60  DATA  AVRIL,"M A I",FIN

200 READ TX1, TX2, TX3, MOIS1$, MOIS2$
210 READ DELI$:IF DELI$="FIN" THEN END
220 GOTO 200
```

L'instruction READ est utilisée pour lire des données à partir d'une instruction DATA. On a vu ci-dessus la définition d'une instruction DATA. L'instruction READ peut être utilisée avec une ou plusieurs variables comme argument, ces variables étant alors séparées par une virgule. Lorsqu'une instruction suivie de plusieurs variables est exécutée, chacune des variables est assignée par la donnée suivante disponible.

Il est important de bien s'assurer que le programme ne va pas lire au-delà de la dernière donnée du dernier DATA, sinon l'erreur 31 est signalée.

RESTORE

<numligne> RESTORE (<numligne>)

```
440 RESTORE
500 RESTORE 20 (pointe sur la ligne 20)
```

Lorsqu'un programme BASIC est lancé, la première exécution d'un READ lit la première donnée du premier DATA, et ainsi de suite. Chaque fois qu'un READ est exécuté, il lit la donnée suivante disponible, en suivant l'ordre des DATA. Lorsqu'un RESTORE est exécuté, cela repositionne la première donnée pour les READ ultérieurs au niveau du premier élément du premier DATA du programme. En d'autres termes, RESTORE positionne le pointeur de "prochaine donnée disponible" au début du groupe de données qui apparaît dans la première instruction DATA du programme.

Si l'on spécifie un numéro de ligne après RESTORE (second exemple) la prochaine lecture de données se fera au premier DATA apparaissant après cette ligne au lieu de se faire au premier DATA du programme.

6.2. BRANCHEMENTS ET SOUS-PROGRAMMES

SUB

Les instructions CALL, SUB et LOCAL sont spécifiques aux SBASIC. Elles permettent de définir des sous-programmes et de leur donner un nom. On pourra ensuite les appeler par l'intermédiaire de ce nom dans une autre partie du programme.

L'instruction SUB permet de définir le nom d'un sous-programme avec la syntaxe suivante :

```
<numligne> SUB <nom-de-sous-programme> <(<var1> <var2>...)>
...Corps du sous programme...
<numligne> RETURN
```

Le nom du sous-programme est un nom de variable qui se construit selon ces règles. La liste (optionnelle) de variables suivant le nom du sous-programme s'appelle la liste d'appel du sous-programme. Elle peut contenir des noms de variables numériques ou alphanumériques sous forme de variables simples ou de tableaux. Dans ce dernier cas, le nom du tableau doit être suivi de ((*)), quel que soit le nombre de dimensions du tableau (1 ou 2 dimensions).

Par exemple :

```
100 SUB ADDITION(NBRELEM% , TB%(*))
110 SOMME=0
115 FOR I%=1 TO NBRELEM%:SOMME=SOMME+TB%(I%):NEXT I%
120 RETURN
```

Une instruction SUB ne doit pas être executée. Seul, un CALL peut l'appeler (voir ci-après).

Les variables et tableaux contenus dans la liste d'appel sont des paramètres formels, qui, au moment de l'exécution, sont remplacés par les valeurs sur lesquelles le sous-programme doit effectivement travailler. Les tableaux formels ne doivent donc pas être dimensionnés à l'intérieur du sous-programme sauf si bien entendu ils ne l'ont pas été dans le programme appellant.

CALL

Permet d'appeler un sous-programme défini par un SUB, avec ou sans passage de paramètres.

La syntaxe en est la suivante :

<numligne> CALL <nom-de-sous-programme> (<var1>, <var2>,...)

La liste de variables figurant dans cette instruction s'appelle la liste d'appel et doit être identique à la liste d'appel du sous-programme appelé : même nombre de variables, même type de variables (réelles, entières, alphanumériques, tableaux), même ordre. Toutefois une variable sans dimension de la liste d'appel peut y être remplacée par un élément d'un tableau du même type ou une expression dont le résultat est du même type que celui de la variable.

Par exemple, utilisons le sous-programme défini ci-dessus dans un programme appelant :

```
10 DIM PUB$(5)
20 FOR I%=1 TO 5:PUB$(I%)="SIROS":NEXT I%
25 N$="SMT"
30 CALL ESSAI(N$,3,PUB$(*))
...

200 SUB ESSAI(A$,N%,TB$(*))......
```

On voit que le tableau PUB est dimensionné dans le programme appelant et non dans le sous-programme.

CALL est terminateur de ligne en Mode direct : aucune instruction qui le suivrait ne serait exécuté.

COMMENT FONCTIONNE LE TRANSFERT DE PARAMETRES ?

Pour les tableaux, variables numériques ou alphanumériques ou éléments de tableaux, le transfert s'effectue par adresse. C'est à dire qu'au moment de l'exécution du sous-programme, les adresses des paramètres sont remplacées par celles des variables de la liste d'appel. Le sous-programme agit alors directement sur les variables contenues dans la liste d'appel qu'il veut modifier.
Quand une expression figure dans la liste d'exécution, elle est évaluée, puis sa valeur est stockée dans une variable tampon transmise au sous-programme.

Si un argument est expressément numérique, l'instruction CALL le convertit dans le type de l'argument formel du sous-programme appelé.

Attention, si l'on veut transmettre à un sous-programme des données qui ne doivent pas être modifiées, il suffit de les mettre entre parenthèses.

Par exemple, en utilisant le sous-programme suivant :

```
100 SUB TEST1(I%,J%)
110 I%=I%+1:J%=J%+1:RETURN
```

Comparer les effets de :

```
10 I%=2:CALL TEST1(I%,J%):PRINT I%,J%:END
```

et de

```
10 I%=2:CALL TEST1((I%),J%):PRINT I%,J%:END
```

LOCAL

Cette instruction s'utilise UNIQUEMENT dans un sous-programme pour masquer les variables qui ont le même nom que dans le programme appelant (création de variables locales).

La syntaxe en est :

<numligne> LOCAL <var1>,<var2>,....

Cette instruction doit être placée juste après la définition du sous-programme "SUB". Elle ne doit être exécutée qu'une fois avant toute référence à l'une quelconque des variables qu'elle contient. Une utilisation ne respectant pas cette règle est susceptible de donner des résultats imprévisibles.

Par exemple :

```
10  I%=10005:CALL DROLE(10):PRINT I%
20  END
100 SUB DROLE(ZEBRE)
110 LOCAL I%
120 I%=ZEBRE:PRINT I%
130 RETURN
```

La variable I% (de valeur 10005) du programme principal ligne 10 n'est pas altérée par l'affectation de la ligne 120. Exécutez une seconde fois ce programme en supprimant la ligne 110 pour en comprendre le rôle.

GOSUB

<numligne> GOSUB <numligne>
<numligne> GOSUB <etiquette>

5 GOSUB 250

Cette instruction transfère le contrôle au sous-programme spécifié à la ligne indiquée. L'exemple appellera le sous-programme de la ligne 250. Tous les sous-programmes doivent s'achever par une instruction RETURN qui rend le contrôle à l'instruction qui suit immédiatement le GOSUB appelant.
Cette instruction peut s'utiliser aussi avec des étiquettes. Une étiquette est définie par le mot réservé LABEL et se construit suivant les mêmes règles qu'une variable de type "virgule flottante".

Dans ce cas, la syntaxe en est :

Ordre d'appel :

```
<numligne1> GOSUB <étiquette>
     ...CORPS DU PROGRAMME...
...END
```

Définition du sous-programme :

```
<numligne2> LABEL <étiquette>
...CORPS ou SOUS-PROGRAMME...
<numligne3> RETURN
```

RETURN

<numligne> RETURN

Exemple : 34 RETURN

RETURN indique qu'il faut revenir au programme appelant alors que l'on se trouvait dans un sous-programme et que l'on désire le quitter. Lorsque le contrôle revient au programme appelant, l'exécution reprend à la première instruction qui suit l'appel. L'instruction qui a appelé le sous-programme est normalement un CALL, un GOSUB ou un ON GOSUB.

GOTO

<numligne> GOTO < numligne>

Exemple : 100 GOTO 50

L'instruction GOTO effectue un branchement simple à la ligne indiquée. Lorsque l'instruction 100 sera atteinte, on sautera à la ligne 50. L'instruction GOTO doit être toujours placée en dernière position dans une ligne comportant plusieurs instructions, parce qu'elle provoque immédiatement le branchement et que les instructions qui la suivent sur la même ligne ne seront jamais exécutées.
Cette instruction peut s'utiliser avec des étiquettes. La syntaxe est alors :

```
<numligne1> GOTO  <étiquette>
            ....
<numligne2> LABEL <étiquette>
```

```
Exemple : 1000 GOTO BRANCHE
          1200 LABEL BRANCHE:PRINT"Suite du Programme"
          1220 END
```

6.3. BRANCHEMENTS CONDITIONNELS

IF THEN

<numligne> IF <expression> THEN <numligne>
ou <instruction> (<:instructions>)

```
500 IF DRAPO%=1 THEN GOTO 110 (ici, GOTO est facultatif)
600 IF A>B AND F>6.0 THEN 300
700 IF X>Y then PRINT"X superieur a Y":PRINT X;Y
```

L'expression est évaluée : si elle est vraie (valeur différente de zéro), on execute l'instruction qui suit le THEN, ou l'on saute au numéro de ligne qui s'y trouve. Si l'expression est fausse, le programme continue en séquence à la ligne suivante. En d'autres termes, cette instruction n'aura aucun effet dans le programme si l'expression est fausse.

```
80 IF A%=0 THEN PRINT "A%=0" : B=1 : GOTO 10
99 IF x=y THEN IF x<z THEN GOTO 40 (ici, le THEN placé après
"X<Z" peut être omis)
```

Dans la ligne 80, si A%=0, nous demandons l'exécution de la suite d'instructions :
"PRINT "A%=0" : B=1 : GOTO 10"
sinon le programme continue en séquence, sur la ligne qui suit.

Dans la ligne 99, on montre que l'on peut effectuer une autre test IF...THEN. Plusieurs instructions de ce type peuvent être imbriquées.

IF THEN ELSE

```
<numligne> IF <expression>
THEN <instruction> (<:instruction>,...)
ELSE <instruction> (<:instruction>,...)
```

```
20 IF PLAF<7410 THEN TAUX=0.055:SAL=SAL-(PLAF*TAUX):GOTO 50
ELSE TAUX=0.047:GOTO SUPLAF
```

Cette instruction est identique à IF THEN. Mais, si l'expression évaluée est fausse, seules les instructions qui suivent ELSE sont exécutées.
Une forme simplifié de cette instruction est :

```
<numligne>IF <expression>THEN <instruction 1> ELSE
                                              <instruction 2>
```

Considérons cette forme et admettons que l'expression ait été évaluée. Si elle est vraie, l'instruction 1 est exécutée. Si elle ne l'est pas, c'est l'instruction 2 qui est exécutée.

Regardons l'exemple. Si l'expression est vraie, on affecte à TAUX la valeur 0,055 et l'on calcule le nouveau salaire. Si l'expression est fausse, le TAUX devient 0,047 et un branchement au Label SUPLAF est effectué. Comme pour les autres instructions conditionnelles, les instructions IF THEN ELSE peuvent être imbriquées.

ON GOSUB

<numligne> ON <expression> GOSUB <liste>

<liste> : liste de numéros de ligne ou d'étiquettes

20 ON AIGUILLAGE% GOSUB 30,40,50,60,ETIQ1,70,ETIQ2

ON GOSUB permet d'appeler un sous-programme parmi plusieurs. L'expression est évaluée et la partie entière du résultat (expression flottante tronquée) détermine où le saut doit s'effectuer. Les numéros de lignes se trouvent en des positions correspondant aux valeurs 1,2,3,4... de l'expression. Ainsi, si AIGUILLAGE% a la valeur 1, le branchement se fera au sous-programme situé à la ligne 30 (1er numéro de la liste). Une valeur 2 pour l'expression appellera le sous-programme situé à la ligne 40 (2ème numéro), et ainsi de suite. Si l'expression évaluée a une valeur inférieure à 1 ou supérieure au nombre de numéros de lignes donnés, un message d'erreur 32 apparaît. L'instruction ON GOSUB doit être la dernière d'une ligne à instructions multiples.

ON GOTO

<numligne> ON <expression> GOTO <liste>

<liste> : liste de numéros de ligne ou d'étiquettes

200 ON SELECTION% GOTO 500,600,700,ETIQ3

Cette instruction fonctionne comme l'instruction ON GOSUB à la différence près qu'elle provoque un branchement à une ligne du programme et non à un sous-programme.
Aucune instruction ne doit suivre ON GOTO dans des lignes à instructions multiples.

ON ERROR GOTO

<numligne> ON ERROR GOTO <numligne>

ON ERROR GOTO 32000

L'instruction ON ERROR GOTO permet à l'utilisateur de contrôler certains types d'erreurs. Tous les numéros d'erreurs peuvent être traités par l'utilisateur, mis à part l'erreur 128. L'instruction ON ERROR GOTO indique à SBASIC le numéro de ligne où il doit se brancher dans le cas où une erreur se produit. C'est le numéro de ligne où commence la routine de traitement d'erreur, soit en 32000 dans le cas de l'exemple. (ATTENTION : ne pas mettre d'étiquette après cette instruction). Voir le chapitre 11. qui lui est réservé "Utilisation de ON ERROR GOTO".

RESUME

<numligne> RESUME <numligne>

RESUME 1000

L'instruction RESUME est utilisée après qu'une routine d'erreur ait été exécutée, pour redonner le contrôle au programme principal au numéro de ligne indiqué (ATTENTION : cette instruction n'accepte pas d'étiquette). Voir la section "Utilisation de ON ERROR GOTO" pour plus de détails.

6.4. ORDRES D'ENTREES-SORTIES

INPUT

<numligne> INPUT ("chaîne";)<liste de variables>

```
100 ON ERROR GOTO 32000
550 INPUT "QUEL EST VOTRE AGE ";A
600 INPUT '"ET ENTREZ VOTRE NOM "';N$
620 PRINT N$;" a";A;"année(s)."
700 INPUT 'DONNEZ-MOI UN ENTIER';I% : PRINT"Merci":END
32000 IF ERL=550 THEN RESUME 550
32010 IF ERL=700 THEN RESUME 700
```

L'instruction INPUT, lorsqu'elle est exécutée, affiche la chaîne de caractères placée entre apostrophes ou entre guillemets qui la suit éventuellement. Elle affiche ensuite un point d'interrogation suivi d'un espace et attend que l'utilisateur entre la ou les données demandées. Dans les instructions demandant plus d'une variable, chaque entrée au clavier doit être séparée de la suivante par une virgule et la dernière doit être suivie d'un retour chariot. Si vous entrez un nombre de données inférieur à ce qui est demandé par l'instruction INPUT, SBASIC répondra au retour chariot par "?", demandant ainsi les données manquantes. Si trop de données ont été entrées, les données supplémentaires sont ignorées.

Notons que l'utilisateur peut répondre à cette instruction par CTRL C. Le programme sera alors interrompu et SBASIC sera prêt à recevoir d'autres commandes. Le programme reprendra son exécution au début de l'instruction INPUT si la commande CONT est tapée.

Si l'utilisateur entre une donnée non conforme au type demandé, alors une erreur est générée, erreur que l'on peut traiter avec ON ERROR GOTO.

INPUT LINE

<Numligne> INPUT LINE <nom de variable de chaîne>

```
10 INPUT LINE A$
20 INPUT LINE BI$(5)
```

L'instruction INPUT LINE (ou INPUTLINE) est utilisée pour entrer une ligne complète dans une variable chaîne. La ligne entière est acceptée, incluant les espaces, les caractères de ponctuation, les apostrophes ou guillemets mais pas le retour chariot. Aucune chaîne texte ne peut être affichée entre guillemets, au préalable comme dans l'instruction INPUT.

PRINT

<numligne> PRINT (liste d'expressions)

```
10 PRINT                        (cause seulement un saut de ligne)
30 PRINT "QUOI"                 (affiche "QUOI" sur l'écran)
40 PRINT "VITESSE=";S
50 PRINT A,B;X;Y
75 PRINT "SOLUTION=";R*PI/1.72 (expression)
```

L'information qui suit l'instruction PRINT peut être affichée telle quelle ou peut être une suite quelconque de constantes, de variables, d'expressions et de chaînes de caractères. Si la suite d'expressions se termine par une virgule ou par un point virgule, le retour chariot à l'affichage ne sera pas réalisé automatiquement ; sinon un saut de ligne sera exécuté après l'affichage des données. Les différents identificateurs de la chaîne d'arguments sont séparés soit par une virgule soit par un point virgule, et l'affichage se comporte différemment suivant les cas.

Le BASIC partage chaque ligne d'édition en 5 champs de 16 caractères chacun. Lorsque les arguments sont séparés par des virgules, la virgule située après un identificateur fera sauter l'affichage au début du champ suivant avant d'afficher l'expression suivante. Ainsi, une virgule fera sauter le début d'affichage des expressions aux tabulations 16, 32, 48 etc... selon le type d'écran ou d'imprimante connecté. Si l'on est positionné en fin d'écran (ou d'imprimante), un saut de ligne s'effectue automatiquement puis l'affichage reprend.

Lorsque les identificateurs sont séparés par des points virgules, les valeurs correspondantes seront affichées de manière contiguë. Il y aura entre eux un ou deux espaces si un ou plusieurs des arguments adjacents sont des nombres. En effet, un nombre est affiché avec un espace au début (sauf s'il est négatif, car dans ce cas, c'est le signe "moins" qui apparaît), puis, un espace de fin. Les chaînes de caractères sont affichées sans espace de début ni de fin.

La bonne utilisation des virgules et des points virgules et la bonne connaissance des formats d'affichage des nombres et des chaînes de caractères vous permettent de bien maîtriser vos formats d'affichage. (voir PRINT USING)

LPRINT

L'instruction LPRINT permet d'écrire simultanément sur écran et sur imprimante. Elle a la même structure que l'instruction PRINT.

PRINT USING

<numligne> PRINT USING <expression chaîne>,<liste à afficher>

```
10 PRINT USING "#####.##", 1234.56
20 PRINT USING "####.#### EST LA RACINE DE ###.##", SQR(X),X
30 PRINT USING A$,I,J,K%
```

PRINT USING est une forme très souple de l'instruction PRINT qui donne à l'utilisateur le contrôle complet du format d'affichage. La chaîne donnée en argument est une image fidèle de la ligne d'édition à l'exception des caractères spéciaux qui sont utilisés pour formater la liste à afficher.

La liste à afficher est la même que celle d'une instruction PRINT, une expression étant séparée de la suivante par une virgule (,) ou un point virgule (;). Les caractères spéciaux de formatage sont les suivants :

LE POINT D'EXCLAMATION

Le point d'exclamation spécifie que seul le premier caractère de chaque champ doit être affiché.

Ex : PRINT USING "! ET ! ET !","AVOIR","BETA","TOUT"

résultat :

A ET B ET T

LE SIGNE BARRE-INVERSE (\ )

Une paire de signes "barres inverses" est utilisée pour délimiter un champ de deux caractères ou davantage. La longueur de ce champ est déterminée par le nombre total de caractères entre les deux signes, plus les signes eux-mêmes. N'importe quel caractère peut être placé entre les deux barres : il sera ignoré. Il est recommandé d'inclure un nombre ou une chaîne de nombres entre les deux barres inverses pour faciliter la documentation.

```
PRINT USING "\34567890123456\","CECI EST UN TEST D'AFFICHAGE"
  Résultat : CECI EST UN TEST
```

SIGNE DIESE

Le signe dièse (#) est utilisé pour définir un champ numérique.

```
PRINT USING "###.##" , 124.555
```

Résultat : 124.56

Si le nombre à afficher est trop grand par rapport au format défini, le signe % précède ce nombre et celui-ci n'est donc plus formatté.

```
PRINT USING "#.##"; 100.31
Résultat : %100.31

PRINT USING "#.#,#.#", 1;2
Résultat : 1.0,2.0
```

Si la partie fractionnaire du nombre ne correspond pas à la définition du champ, le nombre sera arrondi logiquement puis affiché. Si un nombre arrondi est trop grand pour convenir à la définition du format, un "%" sera affiché avant le nombre.

```
PRINT USING "#.# et #.#",1.99,9.99
Résultat : 2.0 et %10.0
```

SIGNE DOLLAR ($)

Le signe ($) est utilisé lorsqu'on veut afficher des montants d'argent en dollars. Le champ est défini de la même manière qu'avec le signe dièse (#) à la différence près que les deux premiers caractères du champ doivent être des signes $.

```
PRINT USING "$$######.##", 123.92
Résultat :    $123.92
```

Notons que les deux signes dollars ajoutent un caractère supplémentaire à la taille du champ numérique pour spécifier le $ de tête. S'il n'y a pas assez de place dans le champ pour placer le $ de tête, le signe % sera affiché comme si le champ dollar n'avait pas été spécifié. Il n'est pas possible de formater un nombre négatif avec un dollar de tête : placer, dans ce cas, un signe négatif après le nombre.

```
A$="$$###.##":PRINT USING A$, 1234
Résultat : % 1234.00
PRINT USING "$$###.##",-1.238
Résultat : % -1.24
```

ASTERISQUE

Le signe astérisque (*) est utilisé lorsque l'on veut remplir les blancs de tête d'un champ numérique avec des astérisques. Cela est particulièrement utile lorsqu'on imprime un champ numérique qui ne doit pas être altéré (par exemple pour des chèques). Le nombre d'astérisques à afficher est mis en tête de chaîne et doit être au moins égal à deux. Chaque astérisque spécifié est un caractère affichable supplémentaire. S'il n'y a pas de place pour au moins un *, le signe % est affiché avant le nombre et le nombre n'est pas formatté.

Le dollar de tête et le champ astérisque ne peuvent pas être utilisés simultanément. Cependant, un signe dollar littéral peut être placé devant le champ astérisque en tant que caractère.

```
PRINT USING "**###.##", 10.2
Résultat : **10.20
PRINT USING "$**###.##", 1.15
$***1.15
```

VIRGULE

La virgule (,) est utilisée pour insérer des virgules dans un champ numérique tous les trois emplacements à gauche du point décimal (à tous les milliers). Si au moins une virgule est présente dans un champ numérique avant le point décimal, les virgules souhaitées seront alors placées de manière appropriée. Une virgule avant le champ numérique ou après le point décimal est considérée comme littérale et sera affichée. Lorsque les virgules du champ numérique font qu'il n'y a plus assez de place, le signe % est affiché, suivi du nombre.

```
PRINT USING "#,###.##",1234.56
Résultat : 1,234.56

PRINT USING "###,###,###", 1E6
Résultat :    1,000,000

PRINT USING "###,###", 1E+6
Résultat : % 1000,000
```

SIGNE MOINS

Le signe moins est utilisé lorsqu'on veut afficher des nombres négatifs et quand un dollar ou un astérisque remplissent le champ de tête. Ce signe est éventuellement ajouté en fin du nombre à afficher.

```
PRINT USING"$$##.## -$$##.##",-10.23;-10.23
Résultat : $10.23- %-10.23
```

SIGNE FLECHE POUR L'EXPOSANT

La flèche (↑)est utilisée pour indiquer un format scientifique pour les champs numériques. Quatre flèches seulement sont permises, elles doivent terminer le champ numérique. Les 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.

```
PRINT USING "#.##########^^^^", SIN(X)
Résultat : 2.55063602616E-01
```

6.5. BOUCLES FOR-NEXT

FOR

<numligne> FOR <variable> =<expression 1> TO <expression 2>
(STEP <expression 3>)

```
70  FOR N%=1 TO 6:PRINT CHR$(7);:NEXT N% (fait 6 fois "BIP")
80  FOR B1=3.2*(X-7) TO 200+Y STEP 5.5
110 FOR I%=10 TO -10 STEP -1
```

Cette instruction permet d'exécuter une certain nombre de fois une ou plusieurs instructions. Lorsque "FOR" est recontré, le SBASIC affecte <expression1> à la <variable> de boucle donnée puis exécute toutes les instructions qui suivent jusqu'à la rencontre du prochain NEXT <variable>.

A ce moment-là, la valeur de <variable> est incrémentée de la valeur contenue dans <expression3> (ou de +1 si <expression3 n'a pas été indiquée), puis la nouvelle valeur de <variable> est comparée à <expression2> :

Si elle devient supérieure à <expression2> alors la boucle "FOR-NEXT" se termine et le BASIC exécute les instructions suivant le "NEXT", sinon il ré-exécute la boucle et ainsi de suite.

L'<expression 3> peut être positive ou négative et ceci a pour effet d'incrémenter ou de décrémenter la valeur de la variable. Si STEP n'est pas précisé, l'incrément est égal à +1. Le test précisé plus haut permet d'exécuter la boucle tant que la valeur de la variable incrémentée de l'<expression 3> est inférieure ou égale à la valeur de <l'expression 2>. Dans tous les cas, la boucle s'effectue au moins une fois.

Les ordres FOR peuvent être imbriqués mais ne doivent pas utiliser les mêmes variables de boucle. Le nombre de niveaux est seulement limité par l'espace mémoire disponible. Les instructions FOR peuvent être interrompues par l'utilisation de l'instruction GOTO située entre les instructions FOR et NEXT, mais ceci est contraire à une bonne clarté de programmation. Bien sûr, l'inverse n'est pas autorisé puisque la <variable> de boucle n'a pas été initialisée, sinon une erreur 62 sera générée.

La valeur de <variable> au sortir de la boucle est indéfinie.

NEXT

<numligne> NEXT <variable>

```
10 FOR I%=1 TO 10
20 PRINT "-";       (imprime 10 fois le caractère Tiret)
30 NEXT I%
```

L'instruction NEXT est uniquement utilisée avec l'ordre FOR. La variable précisée dans l'ordre NEXT est la même que celle de l'ordre FOR auquel il est associé (sinon erreur 62), et sa mention est obligatoire.

Une boucle comportant une <variable> entière s'exécutera beaucoup plus rapidement que la même boucle utilisant une variable de contrôle flottante.

6.6. FIN DE PROGRAMME

END

<numligne> END

```
300 END
```

L'ordre END termine l'exécution d'un programme. Sa présence dans un programme est optionnelle. END peut être placé n'importe où dans un programme. Si un programme est terminé par un END, il ne peut pas redémarrer avec une commande CONT(inue). Utiliser, dans ce cas, l'instruction "STOP".
END referme tous les fichiers.

STOP

<numligne> STOP

50 STOP

Quand une instruction STOP est exécutée, le programme est interrompu, un message est affiché informant l'utilisateur de l'endroit où l'interruption s'est produite. Si l'instruction 50 précédente est exécutée, elle arrête le programme et édite le message :

STOP A LA LIGNE 50

Le programme peut être relancé par la commande CONT et l'exécution redémarre à l'instruction suivant l'ordre STOP, si aucune modification du programme n'a été commencée. L'instruction STOP est similaire à la commande CTRL C.

6.7. LES INSTRUCTIONS DE CREATION ET MODIFICATION DE BUFFERS

Ce paragraphe traite de la manipulation des données dans un tampon ou buffer. Un tampon est une zone réservée en mémoire où l'on peut stocker des informations. Ce tampon peut également être constitué d'une suite de plusieurs zones. Ces dernières sont associées à des variables chaîne à travers lesquelles on peut manipuler les données y résidant. Il y a donc une différence avec une variable chaîne dont la valeur n'a pas de position fixe en mémoire, cette position étant modifiée à chaque nouvelle affectation.

Ces instructions constituent une des nouveautés du SBASIC.

FIELD

L'instruction FIELD est utilisée pour définir le tampon et les noms des zones constituant ce tampon. Chaque zone est identifiée par une variable chaîne. Sa syntaxe est :

<numligne> FIELD <lg1> AS <chaîne 1>,<lg2> AS <chaîne 2>,..

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

crée un tampon en mémoire de longueur 350 octets répartis en trois zone. La première zone est référencée par la variable A$ et a pour longueur 100 octets, la seconde zone est référencée par la variable B$ et a pour longueur 150 octets, enfin la troisième zone est référencée par C$ et a pour longueur 100 octets .

L'instruction FIELD définit donc un tampon utilisable avec la fonction SET, ce qui permet un gain de place et de temps.

Il n'y a pas de mouvement de données entre les variables et le tampon, le contenu des variables étant directement celui des zones du tampon.

L'affectation d'une valeur à une variable déclarée par un FIELD ne peut être faite que par les instructions SET, LSET ou RSET, décrites ci-dessous. Toute affectation d'une valeur par un LET (explicite ou implicite) annule le lien entre la variable et la zone. Par exemple :

```
20 FIELD 50 AS B$
30 B$="CHAINE TEST"
```

la ligne 30 ne place pas la chaîne "CHAINE TEST" dans le tampon mais crée la nouvelle variable B$ qui n'est alors associée à aucun tampon.

LSET et RSET

Le contenu des zones définies par l'instruction FIELD ne peut être modifié que par les instructions d'affectations spéciales RSET, LSET et SET qui ne changent pas l'adresse allouée aux chaînes comme le fait l'instruction d'affectation standard LET (explicite ou implicite). Ces instructions peuvent aussi s'exécuter sur n'importe quelle variable de type chaîne.

La syntaxe des instructions RSET et LSET est la suivante :

```
<numligne> LSET <variable chaîne> = <expression chaîne>
<numligne> RSET <variable chaîne> = <expression chaîne>
```

où <variable chaîne> représente tout nom légal de variable de type chaîne, y compris les variables indicées. Ces instructions rangent le résultat de type chaîne de l'expression à droite du signe égal, dans la zone du tampon associée à la variable chaîne. Avec LSET la chaîne est justifiée à gauche, tronquée si sa longueur est supérieure à celle de la zone, ou complétée avec des blancs si sa longueur est inférieure. RSET justifie la chaîne à droite complétant éventuellement avec des blancs à gauche.

Les deux instructions peuvent être ainsi utilisées avec des chaînes standards.

Exemples d'utilisation :

```
20 FIELD 10 AS A$, 25 AS B$, 10 AS C$
70 LSET A$="VARIABLE A$"
80 LSET B$="NOUVELLE VALEUR POUR B$"
90 RSET C$="NOUVEAU C$ JUSTIFIE A DROITE"
```

Le tampon contient alors l'expression suivante :
VARIABLE ANOUVELLE VALEUR POUR B$ NOUVEAU C$

SET

Cette instruction permet de remplacer une partie d'une chaîne par une autre chaîne donnée. La syntaxe est :

<numligne> SET <expr>, <var> = <chaîne>

où :

- <expr> est une expression numérique entière dont la valeur doit être positive.
- <var> est une variable chaîne.
- <chaîne> une expression chaîne.

Cette instruction a pour effet de remplacer les caractères de <var> situés à partir de celui de rang <expr>, par ceux de <chaîne> et cela jusqu'à épuisement de l'une des deux chaînes.

Par exemple :

```
10 I%=4:A$ ="123456789" : B$ = "--"
20 SET I%+1, A$=B$
30 PRINT A$
```

Sur l'écran, on obtiendra le nouveau contenu de A$:

1234--789

Cette instruction, qui permet de substituer une sous-chaîne de caractères dans une chaîne, est très rapide, 3 à 5 fois plus rapide que l'opération équivalente avec les instructions standards.
Remarque : Il nest pas autorisé d'utiliser cette instruction sur un élément d'un tableau virtuel.

LES INSTRUCTIONS DE CONVERSION CVT

Les instructions SET, LSET et RSET permettent de ranger des données de type chaîne de caractères dans un tampon. Les zones sont nécessairement identifiées par des noms de variable de type chaîne.
Les fonctions de conversion (en fait, de transfert) permettent alors de manipuler dans un tampon des données numériques entières (2 octets) ou réelles (8 octets). Ces fonctions ne réalisent pas à proprement parler, des conversions, puisque les données numériques conservent leur représentation binaire dans le tampon. Il s'agit plutôt d'un artifice permettant d'affecter une valeur "numérique" à une variable de type chaîne, et inversement, ce qui ne nous permet plus de les afficher comme des chaînes de caractères normaux.

Il y a quatre fonctions pour transférer la valeur d'une variable ou d'une constante entière ou réelle dans le tampon et inversement :

```
A$=CVTF$(X)  transfert de réel à chaîne (8 octets)
B$=CVT%$(X%) transfert d'entier à chaîne (2 octets)
X=CVT$F(A$)  transfert de chaîne à réel
X%=CVT$%(B$) transfert de chaîne à entier
```

Il ne faut pas confondre ces fonctions avec les fonctions STR$ et VAL qui effectuent une véritable conversion (binaire/ASCII).

Comme exemple, donnons un programme qui range dans un tampon les éléments d'un tableau de réels :

```
10 DIM A(30)
20 FIELD 31*8 AS A$
30 FOR I%=0 TO 30
40 B$=CVTF$(A(I%))
50 SET 8*I%+1,A$=B$
60 NEXT I%
```

6.8. ORDRES DIVERS

DIM

<numligne> DIM <tableau 1>(,<tableau 2> ...)

```
20 DIM X(20), Y(30), Z(30,40)
```

L'instruction DIM est décrite en détail dans la section 4.4. c'est une instruction de réservation d'espace mémoire pour les variables qui la suivent. L'espace alloué est celui fourni par les assignations. Il est indispensable que tous les tableaux soient dimensionnés avec cette instruction avant d'utiliser leurs éléments dans le programme.

Par exemple, soit un tableau de 100 éléments à deux dimensions (24,3) (rappelons donc que sous SBASIC les indices commencent à zéro). Si le tableau est de type entier, il occupe au total de 25*4*2 = 200 octets d'espace mémoire. S'il est par contre du type réel, il occupera 25*4*8 = 800 octets d'espace mémoire, soit 4 fois plus.

POKE DPOKE

<numligne> POKE <adresse>,<données>
<numligne> DPOKE <adresse>,<données>

```
300 POKE 1000,33
670 L=HEX("CC09"):POKE L,0 (Pause de page "TTYSET PS=N" rapide)
720 DPOKE HEX ("C100"),hex("CC09")
800 DPOKE 8000,0
```

Des données peuvent être fournies aux points de sortie ou aux sous-programmes en langage machine avec les instructions POKE et DPOKE. Il existe deux utilisations fréquentes de ces instructions. POKE stocke l'octet, (DPOKE le double octet) qui est donné dans le second argument à l'"adresse" spécifiée dans le premier argument.

L'adresse et les données utilisées peuvent être des expressions numériques, mais elles sont soumises aux règles suivantes :

0<= adresse<= 32767 (en valeur décimale)
0<= adresse<= 65535 = (HEX ("FFFF")) (quand "HEX" utilisé)

et

0 <= données <= 255 pour POKE (un octet)
0 <= données <= 65535 pour DPOKE (deux octets)

Si ces limites ne sont pas respectées, les erreurs correspondantes sont signalées par SBASIC. Ces instructions permettent de charger directement le contenu d'une mémoire. Elles doivent être utilisées avec beaucoup de précautions.

REM

<numligne> REM (Message...)

```
40 REM Ceci peut être le nom de votre programme
50 REM
```

L'instruction REM est utilisée pour placer remarques et commentaires dans votre programme. Tout ce qui suit l'ordre REM est ignoré, y compris toute instruction qui suit sur la même ligne. Par contre les instructions qui précèdent l'instruction REM sur la même ligne sont exécutées.

DIGITS

<numligne> DIGITS <total> (<fraction>)

```
10 DIGITS 12
11 DIGITS 12,3
```

L'instruction DIGITS précise à SBASIC le nombre de chiffres significatifs l'on veut avoir en édition (PRINT) indépendamment du PRINT USING. Le nombre maximum de chiffres significatifs est 17 et le minimum est 1. Dans plusieurs cas, l'utilisation du maximum 17 peut donner une 17e décimale erronnée, erreur liée à la conversion binaire-décimal. Aussi l'utilisation du maximum 17 n'est-elle pas recommandée.

Le deuxième argument précise combien de chiffres doivent être édités après la virgule. Le nombre <fraction> doit être inférieur ou égal au nombre <total>, sinon une erreur est signalée. Si le nombre <total> précisé est plus petit que le nombre de chiffres significatifs du nombre, l'affichage se fera en notation scientifique.

```
DIGITS 4,3
PRINT PI
```

Résultat :
3.142

```
DIGITS 1
PRINT 10
```

Résultat :
1E+1

NEW n'a pas d'effet sur DIGITS.

SWAP

<numligne> SWAP <variable 1>, <variable2>

```
10 SWAP A,B
>0 SWAP A%(I%), A%(I%+1)
```

L'instruction SWAP permet d'échanger la valeur de 2 variables. Les variables doivent être de même type et ne peuvent pas être éléments de Tableau Virtuel (voir section 12). Son intérêt premier est de gagner du temps lors de tris, (20 à 30 % de gain). Elle est spécialement avantageuse quand on trie des chaînes car elle provoque uniquement l'échange des pointeurs les repérant.

CLEAR

Permet de libérer la place occupée en mémoire par des variables numériques, alphanumériques, et même des tableaux. Pour des variables la syntaxe est :

<numligne> CLEAR <var1>, <var2> ....

Pour des tableaux, la syntaxe est:

<numligne> CLEAR A$(*), AZERTY(*) ,.....

On peut faire figurer dans un même ordre CLEAR variables et tableaux. Après l'exécution d'un tel ordre, la place occupée par les variables de la liste est libérée mais leurs noms se trouvent encore dans la table des symboles. Les tableaux concernés peuvent être alors redimensionnés sans restriction.

CLEAR, utilisé sans arguments libère tout l'espace occupé par les variables, et les remet toutes à zéro. Elle remet toutes les variables chaîne à nul ("") et supprime l'effet d'un éventuel OVERLAY l'ayant précédé.

ATTENTION :

Il ne faut pas faire porter CLEAR sur un argument formel de sous-programme (SUB) : ceci étant imprévisible, il y a risque de provoquer des résulats inattendus.

CURSOR
Permet de positionner le curseur en un point déterminé de l'écran 25 x 80. La syntaxe en est :

<numligne> CURSOR <expr1>, <expr2>

<expr1> et <expr2> sont des expressions numériques représentant respectivement le numéro de ligne (ou "rangée"), et de colonne où l'on désire positionner le curseur.

- <expr1> doit se situer dans la fourchette de valeur 0 à 24,

- <expr2> sà valeur doit être comprise de 1 à 80.

Exemple : CURSOR L%,20 positionne le curseur en rangée L% et en colonne 20.

Si l'une de ces expressions a une valeur non entière, elle est tronquée au préalable à la valeur entière correspondant.
En cas de dépassement des limites indiquées ci-dessus, une erreur est signalée par le SBASIC.
Si <expr1> est égale à 0, alors le prochain "FIN DE LIGNE" ramènera le curseur sur la ligne où il était auparavant positionné, en colonne 1.

EXECUTE
Cette instruction permet d'exécuter une suite d'ordres SBASIC contenus dans une expression chaîne donnée. La syntaxe en est :

<numligne> EXECUTE <expression chaîne>

où <expression chaîne> est une chaîne de caractères contenant la suite ordres SBASIC à exécuter.

ATTENTION : En mode direct, cette instruction est un terminateur de ligne, comme REM. Toute instruction qui la suivrait sur la même ligne serait ignorée.
Cette instruction n'est pas interprêtée par le RUN-TIME "SBRUN".

Exemple :

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

Si lors de l'exécution, l'utilisateur tape

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

la valeur 4 s'affiche à l'écran.

LOAD, BLOAD, et EXEC ne peuvent apparaître dans <expression chaîne>

Cette instruction fait partie des nouveautés de SBASIC et offre un puissant moyen de génération automatique de programmes.

7. FONCTIONS INTRINSEQUES

Les fonctions les plus courantes de SBASIC ont été classées en 5 groupes : Mathématiques, Trigonométriques, Caractères, Entrées-Sorties, Diverses.

Le nom d'une fonction ne peut apparaître que dans une expression, à l'exclusion de tout autre place dans une instruction SBASIC.

Pour utiliser une fonction il suffit d'en connaître le nom, les nombres et types de ses arguments. Ces arguments sont placés derrière le nom de la fonction, entre parenthèses.

Exemple : soit l'instruction :

```
100 Y=SQR(9).
```

Celle-ci, après exécution, affecte la valeur 3 à Y (SQR est la fonction racine carrée).

7.1. FONCTIONS MATHEMATIQUES

EXP(X)

Renvoit l'exponentielle de l'argument donné (e élevé à la puissance x). Ici, "e" est la base des logarithmes naturels et vaut approximativement 2.718281828459045. La valeur maximale de l'argument X est 88.02969193111306. Un argument plus grand que celui-ci produit un "OVERFLOW" et un message d'erreur 102.

LOG(X)

Donne le logarithme naturel (en base "e" ou "logarithme népérien") du nombre X qui doit être positif.
Pour obtenir le logarithme en base B, on utilisera la formule suivante :

```
LOG(X)/LOG(B)
```

Un argument négatif provoque l'erreur 105 .

Exemple : L=LOG(100) renvoit dans L, une valeur d'env. 4,60517

SQR(X)

Fournit la racine carrée de X. Si l'argument X est négatif, l'erreur 107 est signalée.

Exemple : RAC=SQR(5*5) rend la valeur 5 à RAC.

7.2. FONCTIONS TRIGONOMETRIQUES

Pour chacune des fonctions trigonométriques, la précision de la valeur fournie dépend de la grandeur de l'argument. Les fonctions trigonométriques ont une précision minimale de 13,5 décimales.

ATN(X)

Retourne l'arc tangente de X en radians. La valeur retournée est comprise en -pi/2 et pi/2 , où pi/2 est approximativement égal à 1.5707963267948966.

COS(X)

Retourne le cosinus de l'angle X. L'argument X est fourni en radians.

SIN(X)

Retourne le sinus de l'angle X. L'argument X est fourni en radians.
Exemple : PRINT SIN(1.2) affiche 0.932 etc...

TAN(X)

Retourne la tangente de l'angle X. L'argument X est fourni en radians.
Un résultat trop grand produit une erreur "OVERFLOW" No 104.
Exemple : PRINT TAN(1) affiche 1.5574 etc...

Les fonctions trigonométriques acceptent les réels comme arguments, sans limite, mais elles fonctionnent de manière plus rapide lorsque l'argument est compris entre -32000 et +32000.

7.3. FONCTIONS CHAINES DE CARACTERES

ASC(X$)

L'argument X$ est une expression chaîne. La valeur retournée par la fonction sera la valeur numérique ASCII du 1er caractère de la chaîne. Zéro sera retourné si l'argument est une chaîne nulle .
Exemple : PRINT ASC("AZERTY") donne 65.

CHR$(I%)

Cette fonction fournit un simple caractère ASCII (une chaîne d'un caractère) qui est le caractère dont le code ASCII est I%.
L'argument I% doit être un nombre compris entre 0 et 255.
Exemple : PRINT CHR$(66) rend le caractère "B".

HEX(X$)

La fonction HEX convertit une chaîne de caractères hexadécimaux en valeur décimale équivalente.
Exemple : A$="100":PRINT HEX(A$) affiche la valeur 256.

INCH$(I%)

La fonction INCH$(I%) entre un caractère depuis le canal I%, qui peut être le clavier (I%=0), ou un fichier séquentiel (I%<>0 et le fichier doit alors être ouvert). L'argument I% précise le canal interne du fichier. Le canal 0 est celui du terminal utilisateur (clavier).
Sur le canal 0 l'entrée d'un caractère se fait alors sans écho, (c'est-à-dire que les caractères entrés ne sont pas affichés) contrairement à ce qui se passe avec l'instruction INPUT. Tous les caractères sont acceptés par cette fonction.
Exemple : REPONSE$=INCH$(0) attend la frappe d'une touche et renvoit la valeur dans REPONSE$ sans l'afficher.

INCH$(-1)

Cette fonction permet de savoir si une touche a été frappée au clavier. Dans l'affirmative, elle retourne le caractère frappé au clavier, sinon elle retourne une chaîne d'un caractère dont la valeur ASCII est égale à 0 soit CHR$(0).

Exemple :

```
10 I=0
20 IF INCH$(-1)=CHR$(0) THEN I=I+1 : GOTO 20
30 PRINT I;
```

Ce programme ne s'arrêtera que lorsque l'utilisateur enfoncera une touche quelconque puis affichera le nombre d'explorations de clavier effectuées.

INSTR(I%,CH$,S$)

La fonction INSTR recherche une sous-chaîne S$ dans la chaîne de caractère CH$. Le premier argument précise à partir de quel caractère de la chaîne CH$ où doit commencer la recherche. INSTR retourne une valeur entière précisant à quel caractère la sous-chaîne S$ débute dans CH$. Si la sous-chaîne n'est pas contenue dans CH$, la valeur zéro est retournée.

Exemple : PRINT INSTR(1,"MICRO","CR") affiche 3.

LEFT$(X$,I%)

La fonction caractère "LEFT$" renvoit une chaîne constituée des I% octets les plus à gauche de X$. La valeur de I% doit être positive et inférieure à 32767 sinon une erreur est signalée.

```
Exemple : D$="SBASIC GOUPIL":A$=LEFT$(D$,6)
          A$ contient le mot "SBASIC".
```

LEN(X$)

La fonction LEN(X$) retourne la longueur de la chaine X$ en nombre d'octets. Tous les octets de la chaîne sont comptés, même les espaces et les octets non affichables.

Exemple : DOC$="DOCUMENTATION":PRINT LEN(DOC$) affiche 13.

MID$(X$,I%)

Cette fonction retourne une partie de la chaîne X$. La chaîne retournée commence à la position I% dans la chaîne X$ et comprend tous les octets jusqu'à la fin de cette chaîne. I% doit être un nombre positif inférieur à 32767 sinon une erreur est signalée.

Exemple : PRINT MID$("MICRO-ORDINATEUR",7) renvoit la sous-chaîne "ORDINATEUR".

MID$(X$,I%,J%)

Cette fonction à trois arguments est la même que la précédente à la différence que la sous-chaîne générée comprend J% octets à partir de la position I%.
Exemple : A$="surestimer":PRINT MID$(A$,4,6) rend "estime".

RIGHT$(X$,I%)

Retourne les I% octets les plus à droite de la chaîne X$. Si la valeur de I% est plus grande ou égale à la longueur de la chaîne X$, la chaîne complète est retournée. L'erreur 74 est signalée si la valeur de I% est négative, ou si elle est plus grande que 32767.
Exemple : EX$="SBASIC":PRINT RIGHT$(EX$,3) affiche SIC.

STR$(X)

Cette fonction retourne une chaîne de caractères qui représente l'expression numérique de X. En d'autres termes, elle prend un nombre et le transforme en chaîne de caractères numériques. La chaîne est construite de manière identique à une édition par un ordre PRINT, avec les espaces, le point virgule et le signe moins. C'est l'inverse de la fonction VAL(X$).
Cette fonction utilise les paramètres définis par l'instruction DIGITS.
STR$ et l'affichage des nombres : le passage en notation scientifique ne se produit que lorsque le nombre de chiffres affichés dépasse celui défini par l'instruction DIGITS.

Exemple : NB=123.145:X$=STR$(NB), alors X$=" 123.145 ".

STRING$(CH$, N%)

Cette fonction envoie une chaîne de caractères formée de N% fois la chaîne CH$.

Exemple : PRINT STRING$("TI",2); STRING$("CA",2)
rend "TITICACA" !

VAL(X$)

VAL(X$) convertit une chaîne de caractères numériques (composée entièrement de nombres, de signe + ou -, et du point décimal positionnés convenablement) en une valeur numérique. C'est l'inverse de la fonction STR$(X).
Si un caractère non numérique est rencontré alors tout ce qui suit n'est pas pris en compte.

Exemple : R$="520.50":A=VAL(R$) retourne dans A la valeur correspondante.
PRINT VAL("3 XY-5") affiche simplement 3.

LTRIM$(A$) et RTRIM$(A$)

Suppression des espaces de gauche et de droite respectivement.

Ces fonctions permettent de supprimer les caractères blancs de début (ou à gauche) de la chaîne A$ avec LTRIM$(A$), et ceux de fin (ou à droite) de la chaîne A$ pour RTRIM$(A$).

Exemples :

```
10 A$="   LE ROI BOIT   "
20 L$=LTRIM$(A$):R$=RTRIM$(A$):
30 PRINT "*";L$;"*" : PRINT "*";R$;"*"
```

affiche *LE ROI BOIT * puis
* LE ROI BOIT* .

Pour des raisons de compatibilité, les anciennes fonctions LTRM$ et RTRM$ sont acceptées, mais l'interprêteur les transforme directement dans le nouveau format, soit : LTRM$ équivaut à RTRIM$, et RTRM$ à LTRIM$.

7.4. ENTREES-SORTIES

PEEK(I) et DPEEK(I)

PEEK(I) et DPEEK(I) retourne le contenu de la mémoire située à l'adresse I (où I est une valeur flottante). La valeur retournée sera supérieure ou égale à 0 et inférieure ou égale à 255 dans le cas de PEEK, et, >= 0 et <= 65535 dans le cas de DPEEK. Si la valeur de l'argument est négatif ou plus grand que 65535, une erreur 75 sera signalée.

POS(I%)

La valeur retournée est la position de la colonne courante du canal I%, la numérotation commençant à zéro. Ainsi, lorsque la valeur retournée est zéro, elle indique la première colonne de la ligne).

Exemple : PRINT POS(0) affiche le numéro de la colonne où se situe le curseur.

SPC(I%)

Cette fonction doit être utilisée uniquement dans un ordre PRINT. Elle permet un espacement de I% caractères à l'édition. SBASIC signale une erreur 74 si la valeur de I% est négative ou plus grande que 255.

Exemple : PRINT SPC(80) affiche 80 espaces.

TAB(I%)

Cette fonction doit être utilisée uniquement dans un ordre PRINT. Elle déplace le curseur à la colonne I% sur l'écran (colonnes numérotées à partir de 0), ou déplace la tête d'écriture de l'imprimante à cette même position. Si le curseur est au-delà de cette position, l'instruction est ignorée. En général, l'argument doit être positif et inférieur ou égal à 255.

L'utilisateur doit être prudent s'il affiche des informations contenant des caractères accentués codés sur 2 ou 3 octets.

7.5. FONCTIONS DIVERSES

ABS(X)

Cette fonction fournit la valeur absolue de X. Si X est positif, la valeur retournée est X, et s'il est négatif, la valeur retournée est -X.

INT(X)

La valeur retournée est la partie entière de X.
Exemples :

```
INT(70)   rend  70.
INT(-5.1) rend  -6.
INT(5.7)  rend  5.
```

PTR(<nom de variable>)

La fonction PTR retourne l'adresse de la variable donnée en argument. Si la variable est de type numérique, l'adresse retournée sera l'adresse du premier octet de ce nombre.
Les nombres du type flottant sont mémorisés sur 8 octets. Les sept premiers fournissent la mantisse, et le dernier l'exposant.
Les nombres du type entier sont représentés sur deux octets.
Si l'argument est une variable chaîne de caractères, la valeur retournée est l'adresse (2 octets) de sa description.
Ce descripteur comprend quatre octets :
les 2 premiers contiennent l'adresse réelle de la chaîne, les deux suivants fournissent sa longueur.

Exemple : PRINT PTR(A) rend l'adresse où est stocké le contenu de la variable A.

PRINT PTR(CH$) donne l'adresse du descripteur de la variable CH$. Admettons que cette fonction nous ait donné 23050, alors : PRINT DPEEK(23050) rend l'adresse de début de CH$ et DPEEK(23052) le nombre d'octets composant cette chaîne.

PI

La valeur retournée de PI est 3.1415926535897933.
Exemple : C=10*PI donne à C la valeur 31.4159 etc...

RND(X)

La fonction retourne un nombre aléatoire (pris au hasard) compris entre 0 et 1. L'utilisateur peut utiliser cette fonction pour générer des nombres aléatoires, compris entre deux limites, par utilisation de la formule :

Nombre aléatoire = (PG-PP) * RND + PP

où PG est le plus grand nombre et PP le plus petit nombre souhaités. Le résultat de cette formule rend un nombre aléatoire compris entre PG et PP.

L'argument X permet de créer le nombre aléatoire suivant les règles suivantes :
X<0 : la fonction fournit la même valeur chaque fois qu'elle est appelée pour la même valeur de X.
X=0 : Permet à la fonction de créer un nombre aléatoire nouveau chaque fois qu'elle est appelée. C'est l'argument usuel d'utilisation.
X>0 : Permet de retourner le dernier nombre aléatoire créé.
Exemple : Générons une suite de 10 nombres compris entre 0 et 99 inclus :

```
100 I=RND(-45.46):REM "-45.46" peut être tout autre nombre
                  négatif, il permet d'initialiser le
                  générateur de nombres aléatoires (RANDOMIZE).
110 FOR I%=1 to 10 : PRINT INT(RND(0)*100); : NEXT I%
```

SGN(X)

C'est la fonction signe. Elle retourne "1" si X est plus grand que zéro, zéro si X est égal à zéro ou "-1" un si X est négatif.
Exemple : PRINT SGN(-45);SGN(45.46) retourne -1 1.

FRE(0)

FRE(0) retourne le nombre d'octets encore disponibles dans la mémoire. L'argument numérique est sans signification, mais il est nécessaire syntaxiquement.
Exemple : PRINT FRE(0) affiche la taille mémoire disponible.

DATE$

DATE$ fournit la date du système d'exploitation FLEX-9, sous la forme d'une chaîne de 9 caractères selon le format : JJ-MMM-AA où :

"JJ" représente le jour sur deux chiffres ;
"MMM" le mois sur trois lettres et
"AA" l'année sur deux chiffres.
Exemple : D$=DATE$:PRINT D$ renvoit dans D$ la date du système comme : "02-NOV-83" pour le 2 nov. 1983.

COMMANDES, INSTRUCTIONS ET FONCTIONS SPECIFIQUES AU GRAPHISME COULEUR ET A LA COMMUNICATION

8.1. LE GRAPHISME COULEUR

Les commandes suivantes permettent de réaliser des programmes SBASIC graphiques (il vous faudra, bien sûr, posséder la carte graphique couleur de GOUPIL qui permet un affichage de 8 niveaux de couleur avec une définition de 512 x 256 points).

GR

Mode Graphique : 256x256

<numligne> GR

L'instruction GR initialise l'affichage du graphique sur l'écran couleur et l'affichage du texte sur l'écran noir et blanc ainsi que la carte graphique. S'il n'y a qu'une vidéo noir et blanc, le graphique ne s'affichera pas. S'il n'y a qu'une vidéo couleur, le texte ne s'affichera pas.

L'ordre GR provoque un effacement de l'écran couleur sur fond de la dernière couleur spécifiée (la couleur de fond est noire à l'initialisation).

Dans le graphique, pas d'argument par défaut.

HGR

Mode Graphique : haute résolution

<numligne> HGR

Permet d'activer l'écran graphique en mode 512 x 256 points. Produit les mêmes effets que l'instruction précédente.

TEXT

<numligne> TEXT

L'instruction TEXT provoque en SBASIC la PERMUTATION des écrans graphiques et texte, ce qui permet de visualiser un graphique sur l'écran texte standard. Cette instruction agit comme une bascule.

S'il n'y a que l'écran couleur (télévision par exemple), TEXT permet d'avoir alternativement des graphiques et du texte sur l'écran noir et blanc.

SETCOLOR

La carte GRAPHIQUE COULEUR permet de définir 255 couleurs physiques à partir des trois couleurs fondamentales, ROUGE, VERT, et BLEU (R, V, B).

Cependant, il ne peut exister simultanément que 8 couleurs sur un écran à un instant donné, couleurs que l'on appellera couleurs logiques. On peut redéfinir ces couleurs logiques à tout instant, et c'est l'ordre SETCOLOR qui remplira cette fonction.

L'ordre SETCOLOR permet de définir les couleurs logiques comme combinaisons des trois couleurs fondamentales, ROUGE, VERT, et BLEU.

La syntaxe en est :

```
SETCOLOR <couleur-logique>,<coefficient-rouge>,
         <coefficient-vert>,<coefficient-bleu>
```

- <couleur-logique> est une expression dont la valeur doit être comprise entre 0 et 7, valeur qui sera définie dans l'ordre COLOR pour la couleur courante ;

- <coefficient-> est une expression dont la valeur doit être comprise entre 0 et 7. Cette valeur est proportionnelle à l'intensité de la composante correspondante.

Toute autre valeur provoque l'émission d'un message d'erreur approprié.

Exemples :

```
10 R% = 6
20 V% = 5
30 B% = 3
40 SETCOLOR 3, R%, V%, B%
```

met la couleur logique No 3 à un jaune vif. En effet, combiner du rouge et du vert intense avec du bleu moyen donne un jaune vif.

```
120 SETCOLOR 3,,,7
```

met le coefficent bleu à 7 pour la couleur 3 de l'exemple précédent, les autres coefficients restant inchangés.

Il faut noter que tout ce qui a été précédemment tracé ou écrit dans une certaine couleur logique prend une nouvelle teinte chaque fois que l'on re-définit cette couleur logique par un ordre SETCOLOR.

A la mise sous tension, les couleurs logiques ont les valeurs par défaut suivantes :

COULEUR	- R -	- V -	- B -	
0	0	0	0	NOIR
1	7	0	0	ROUGE
2	0	7	0	VERT
3	7	7	0	JAUNE
4	0	0	7	BLEU
5	7	0	7	MAGENTA
6	0	7	7	CYAN
7	7	7	7	BLANC

SETBLINK

Cette instruction est tout à fait similaire à SETCOLOR.

Elle permet de définir une couleur logique secondaire à celle définie par SETCOLOR.

Ces deux couleurs alterneront par clignotement successif, au rythme de 0,4 seconde chacune.

La syntaxe en est :

SETBLINK <couleur-logique>,<coefficient-rouge>,
<coefficient-vert>,<coefficient-bleu>

Pour annuler le clignotement, il faut re-définir la couleur logique par SETBLINK, avec les mêmes valeurs de paramètres que SETCOLOR.

Il faut noter que tout ce qui a été précédemment tracé ou écrit dans une certaine <couleur-logique>, se met à clignoter dès que l'on re-définit cette couleur logique par un ordre SETBLINK.

COLOR

<numligne> COLOR N

La variable N peut prendre une valeur comprise entre 0 et 7 (8 couleurs sont donc possibles).
N= : COULEUR
=0 Noir
=1 Rouge
=2 Vert
=3 Jaune
=4 Bleu
=5 Magenta
=6 Cyan
=7 Blanc

Après cette instruction, tous les points apparaissent sur l'écran dans la couleur ainsi donnée. Si N est supérieur à 7, l'erreur 110 est générée.

MASK

<numligne> MASK (R,V,B)

L'affichage couleur de l'écran est constitué de la superposition des trois plans de couleurs fondamentales (rouge, vert, bleu).

L'instruction MASK permet de masquer les plans choisis.

0 : non-masqué
1 : masqué

Exemple : masquage du plan vert :

```
MASK (0,1,0)
```

CLRG

Effacement de l'écran Graphique avec la couleur courante.

Exemple :

```
10 SETCOLOR 3,7,0,0:REM  met la couleur logique 3 à rouge vif
20 COLOR 3:REM  sélectionne la couleur 3 comme couleur courante
30 CLRG:REM          l'écran s'allume uniformément en rouge vif
```

Il faut nqter que si l'on redéfinit la couleur logique 3 par l'ordre SETCOLOR, la couleur physique de fond d'écran est changée dès exécution de cet ordre.

PLOT et PLOT TO

<numligne> PLOT X, Y

cet ordre affiche avec la couleur courante le point de coordonnées (X,Y) sur la matrice 512 x 256.

<numligne> PLOT TO X,Y

Cet ordre affiche un vecteur compris entre le point courant et le point de coordonnée X,Y.

<numligne> PLOT X,Y TO X1,Y1

Cet ordre affiche un segment de droite compris entre les points de coordonnée X,Y et le point de coordonnée X1,Y1 sur la matrice 255 x 255.

Exemple :

```
10 GR
20 COLOR 4
30 PLOT 1,1 TO 256, 256
40 PRINT "PAPA"
50 END
```

Ce programme tracera une diagonale bleue sur l'écran couleur et affichera "PAPA" sur l'écran vidéo normal.

MOVE

La syntaxe de cette instruction est :

MOVE <X>,<Y>

où <X> et <Y> sont définis comme précédemment.

Cette instruction permet de déplacer le point courant aux coordonnées <X> et <Y> sans rien tracer.

Exemple :

```
 10 COLOR 10
 20 GR
 30 CLRG
 40 COLOR 1
 50 ZERO% = 128
 60 MOVE 0,ZERO%
 70 FOR I%=0 TO 100
 80 X = I%*PI/100
 90 PLOT TO I%,ZERO%+100*SIN(X**2)
100 NEXT I%
110 END
```

Ce programme permet de tracer par segments successifs, la courbe Y=SIN(X**2).

DASH

La syntaxe de cette instruction est :

DASH <nature-du-tracé>

où <nature-du-tracé> est une expression dont la valeur est comprise entre 0 et 3.

Cette instruction précise la nature du tracé.

<nature-du-tracé>	:	0	trait continu
	:	1	pointillé fin
	:	2	pointillé gros
	:	3	mixte (trait et point)

La valeur initiale de <nature-du-tracé> est 0.

La valeur définissant la nature du trait reste inchangée entre deux exécutions de DASH.

SYMBOL

Cette instruction permet d'afficher sur l'écran graphique, des textes de taille et de forme variables.

La syntaxe en est :

SYMBOL <X>,<Y>,<chaîne>[,<largeur-des-caractères>,
<hauteur-des-caractères>,<attributs>]

<X> et <Y> sont les coordonnées du point de début d'écriture, en bas et à gauche du premier caractère.

<chaîne> contient les caractères à tracer (voir la définition dans le Manuel de Présentation et d'Installation).

<largeur-des-caractères> et <longueur-des-caractères> doivent être des expressions comprises entre 1 et 16 en mode 512 x 256 (Mode HGR), ou entre 1 et 8 en mode 256 x 256 (Mode GR).

Ce sont les tailles d'un point élémentaire définissant les caractères tracés dans une matrice 6 x 8.

Exemple :

Si
<largeur-des-caractères> vaut 4 et

<hauteur-des-caractères> vaut 7

alors les caractères du texte seront tracés dans un rectangle 24 x 56 points et définis par des blocs de 4 x 7 points.

<attribut> est une expression dont la valeur est comprise entre 0 et 3.

<attribut> : 0 tracé horizontal, caractères droits
1 tracé horizontal, caractères penchés
2 tracé vertical, caractères droits
3 tracé vertical, caractères penchés

POINT

Cette fonction retourne un entier compris entre 0 et 7 donnant la couleur logique du point désigné. Sa syntaxe est :

POINT (<X>, <Y>)

où <X> et <Y> sont des expressions donnant les coordonnées comprises entre -32768 et 32767 désignant le point.

Si le point est hors de l'écran, la valeur 0 est retournée.

Exemple :

100 I% = POINT(KX%,KY%) rend la couleur du point de coordonnée KX% et KY%.

ARC

Cette instruction permet de tracer un arc de cercle ou un cercle complet.

La syntaxe en est

ARC <X-centre>,<Y-centre>[,<angle>][,<X-départ>,<Y-départ>]

<X-centre> et <Y-centre> sont les coordonnées du centre du cercle. Ce sont des expressions qui doivent être comprises entre -32768 et 32767. Il s'agit d'un point virtuel pouvant se trouver hors de l'écran.

<angle> est une expression qui donne l'angle de l'arc de cercle en degrés.

Si la valeur de l'angle est omise et si elle est supérieure à 2 * PI, ou inférieure à -2 * PI, le cercle complet est tracé.

Le sens du tracé est le sens trigonométrique positif (c'est à dire le sens contraire des aiguilles d'une montre) si l'angle est positif, et dans le sens trigonométrique négatif si l'angle est négatif.

<X-départ> et <Y-départ> donnent les coordonnées au point de déaprt et obéissent aux mêmes règles de définition que <X-centre> et [Y-centre>.

S'il est omis, le point de départ est le point courant. Le dernier point tracé de l'arc devient le point courant.

Exemple 1 :

```
10 COLOR 0
20 GR
30 CLRG
40 COLOR 1
50 ARC 128,128,,228,128
```

trace le cercle qui a pour centre le milieu de l'écran et pour rayon 100.

Exemple 2 :

```
10 COLOR 0
20 GR
30 CLRG
40 COLOR 1
50 ARC 128,128,PI/2,228,128
```

trace le quart de cercle de rayon 100 du premier cadran qui a pour centre le milieu de l'écran.

FILL

Deux syntaxes sont autorisées :

FILL <X>,<Y>

où <X> et <Y> sont les coordonnées d'un point appartenant à une aire colorée. Cette aire prend par cette instruction la couleur courante.

et :

FILL <X>,<Y>,<couleur>

où <X> et <Y> sont les coordonnées d'un point appartenant à une aire délimitée par un contour de couleur <couleur>. Cette aire peut être elle-même composée de zones de plusieurs couleurs.

La commande FILL permet de remplir toutes ces zones avec la couleur courante.

DRAW

Permet de dessiner de petits motifs répétitifs.

La syntaxe est :

DRAW <X>,<Y>,<chaîne>(,<rotation>)

20 DRAW 128,128,"1Sd5Gf2Rt",2

- <X> et <Y> sont les coordonnées du point de départ, l'origine du dessin.

- <chaîne> = ...

  <chaîne> définit le tracé du dessin par une succession de segments de droite :

 = (<chiffre>)<MAJUSCULE><minuscule>

<chiffre> compris entre 0 et 8
définit la couleur du tracé (0-7)
sachant que 8 supprime le tracé.
Par défaut, dernière couleur demandée.

<MAJUSCULE> compris entre A et Z
définit le déplacement relatif en X par rapport au point courant (voir tableau des correspondances).

<minuscule> compris entre a et z
définit le déplacement relatif en Y par rapport au point courant (voir tableau des correspondances).

- <rotation> compris entre 0 et 7
effectue une rotation du dessin de <rotation> * 45° par rapport au point origine.
Par défaut la rotation est nulle.

CORRESPONDANCE LETTRE/DEPLACEMENTS 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	O	P	Q	R	S	T	U	V	W	X	Y	Z
0	1	2	3	4	5	6	7	8	9	10	11	12

CORRESPONDANCES CHIFFRE/ANGLE DE ROTATION

0	1	2	3	4	5	6	7
0	45	90	135	180	225	270	315

Exemple :

```
10 A$="8An7NzAa2ZaPu"
20 GR:COLOR 0 : CLRG : COLOR 7
30 PLOT 128,128 : REM marque l'origine
40 DRAW 128,128,A$,1
```

WINDOW

Cette instruction permet de définir la fenêtre rectangulaire de tracé graphique.

La syntaxe est :
WINDOW <X1>,<Y1>,<X2>,<Y2>

où <X1>,<Y1> et <X2>,<Y2> sont les coodonnées de deux points diagonaux sur l'écran graphique. Ce sont des expressions dont la valeur doit être comprise entre -32768 et 32767.

Si ces valeurs dépassent les limites de l'écran, c'est-à-dire en dehors de :
- 0 à 255 pour les X en mode GR,
- 0 à 511 en mode HGR,
- 0 à 255 pour les Y
  elles seront alors tronquées à ces limites.

L'instruction WINDOW s'applique aux ordres suivants :
PLOT, SYMBOL, DRAW, ARC et FILL.
Tout tracé qui sort de la fenêtre sera ignoré et n'apparaîtra pas à l'écran.
Les valeurs par défaut sont :

WINDOW 0,0,511,255 dans le mode HGR et
WINDOW 0,0,255,255 dans le mode GR.

PEN

Cette instruction permet d'activer le photostyle (ou Light-Pen), et arrête le programme en cours jusqu'à ce que l'on ait validé la position du photostyle en l'appuyant à l'endroit désiré.
Cet ordre agit comme un ordre INPUT, à la différence que la lecture ne se fait pas au clavier mais sur le photostyle.

La syntaxe en est la suivante :

PEN <option>

où <option> est une expression pouvant prendre les valeurs 0, 1 ou 2. L'option par défaut est 0.

VALEUR 0

Le programme est mis en attente, et dès qu'un point suffisamment lumineux est désigné sur l'écran graphique, les variables XPEN et YPEN sont mises à jour avec les valeurs des coordonnées du point désigné. L'exécution du programme reprend à l'instruction suivant PEN.

VALEUR 1

L'exécution du programme n'est pas interrompue. Les variables XPEN et YPEN sont mises à jour avec les coordonnées du point désigné, s'il était suffisamment lumineux au moment de l'exécution de PEN. Dans le cas où le point n'était pas assez lumineux, XPEN et YPEN sont mis à -1.

VALEUR 2

Cette option agit comme la précédente, à la différence près qu'il y a émission d'un flash sur l'écran graphique couleur, de façon à illuminer tous les points quand l'instruction PEN est exécutée.

XPEN et YPEN

Ce sont des variables systèmes (au même titre que ERL, ERR ou PI) mais elles rendent la position du photostyle en coordonnée horizontale (XPEN) et verticale (YPEN) au moment de la dernière validation.

On les utilisera comme des valeurs entières dans des expressions, mais elles ne peuvent pas figurer dans une instruction d'assignation (LET explicite ou implicite).

Exemple :

```
100 PEN
110 IF (XPEN>=10 AND XPEN<=21) AND (YPEN>10 AND
YPEN<=21) THEN 200
130 PRINT "Vous n'avez pas désigné le bon objet, recommencez"
140 GOTO 100
200 PRINT "Réponse correcte"
```

La ligne 100 met le programme en attente de validation. Une fois le photostyle appuyé, la ligne 110 est exécutée. Elle vérifie que le point de coordonnées XPEN,YPEN se trouve bien à l'intérieur d'un carré de côté 11 dont le coin "sud-ouest" se trouve au point de coordonnées 10,10.

AFFICHAGE SIMULTANE DE TEXTES ET DE DESSINS SUR L'ECRAN GRAPHIQUE COULEUR

GOUPIL permet d'afficher simultanément sur l'écran graphique couleur des graphiques et des textes aux normes VIDEOTEX, pour réaliser par exemple des jeux avec dialogues.

Il faut pour cela définir une fenêtre virtuelle.
Exemple :

```
100 PORT 4 (aiguille les sorties sur l'écran graphique couleur)
110 PRINT CHR$(27);CHR$(91);"20";CHR$(59);"25";CHR$(85)
120 PRINT "TEXTE GRAPHIQUE"
```

définit une fenêtre textuelle comprise entre les "lignes" 20 et 25 de l'écran graphique couleur (l'écran est découpé en 25 lignes de 40 caractères).

La fenêtre par défaut recouvre tout l'écran (de la ligne 01 à la ligne 25). Les débordements par rapport aux limites de l'écran sont tronqués à ces limites.

On peut écrire dans la fenêtre textuelle au moyen d'ordres PRINT, tout comme sur l'écran vidéo 25 x 80 ou l'écran VIDEOTEX (voir le paragraphe 1.2.5 du manuel de présentation et d'installation).

Attention : L'instruction PORT 4 n'est pas en fonction sur toutes les différentes versions de GOUPIL.

On peut ainsi mixer sur l'écran graphique le mode dessin et le mode texte plus facile à manipuler par PRINT que par SYMBOL

Les fenêtres textuelles et les fenêtres graphiques étant définies de manière indépendante, il est possible que les textes et les dessins se chevauchent, ce qui est parfois utile. Si l'on ne souhaite pas un tel chevauchement, il faut définir les fenêtres de sorte qu'elles ne se recouvrent pas.

Il faut noter que l'ordre Graphique "SYMBOL" permet d'écrire des textes en mode GRAPHIQUE, contrairement à l'ordre PRINT qui affiche sur toute une ligne et agit en mode TEXTE sur l'écran graphique,. dans une fenêtre TEXTUELLE.

Les normes d'affichage de textes en graphique couleur sont définies en Annexe E du manuel de présentation et d'installation.

PDL

Cette fonction retourne un nombre compris entre 0 et 255 qui donne la position de la Manette de jeu (Paddle) correspondante.

La syntaxe est :

PDL (<numéro-paddle>)

où <numéro-paddle> est une expression dont la valeur est comprise entre 0 et 3.

- 0 retourne la coordonnée X de la manette 0 ;
- 1 retourne la coordonnée Y de la manette 0 ;
- 2 retourne la coordonnée X de la manette 1 et
- 3 retourne la coordonnée Y de la manette 1.

8.2. LA COMMUNICATION

PORT

SBASIC permet la communication avec les différents organes intérieurs (carte CONTROLEUR VIDEO 25 x 80, EXTENSIONS ENTREES-SORTIES, GRAPHIQUE COULEUR) ou extérieurs (imprimantes, modems, etc.) de GOUPIL.

La syntaxe est la suivante

<numligne> PORT <numéro-de-port>

Cette instruction permet d'aiguiller les informations en entrée (ordre INPUT) ou en sortie (ordre PRINT), suivant la valeur de <numéro-de-port>.

<numéro-de-port> est une expression numérique qui donne le numéro du canal choisi et dont seule la partie entière est conservée.

Les <numéros-de-port> se répartissent en trois classes et ont la signification suivante :

- entre 0 et 9 : aiguillage des ordres PRINT sur les périphériques de sortie
- entre 10 et 19 : aiguillage des ordres INPUT sur la périphériques d'entrée
- entre 20 et 29 : aiguillage des ordres PRINT vers les périphériques d'impression.

L'ordre PORT agit, selon la classe à laquelle il apartient, sur les ordres PRINT et INPUT qui sont exécutés après lui. Par exemple, si nous supposons une configuration GOUPIL comprenant un écran GRAPHIQUE COULEUR, un écran VIDEO 25 x 80, et une imprimante :

```
10  PRINT "HELLO"    affiche HELLO sur l'écran 25 x 80 (PORT 0
                     par défaut)
20  PORT 1           aiguille les PRINT suivants sur
                     l'imprimante
30  PRINT "Voilà"    édite "Voilà" sur l'imprimante
40  PRINT "mille"    édite "mille" sur l'imprimante
50  GR               connecte l'écran graphique
60  PORT 4           aiguille les PRINT vers l'écran graphique
                     couleur
70  PRINT "roses"    édite "roses" sur l'écran graphique couleur
80  PORT 0           permet de revenir à l'affichage par défaut
90  PRINT "amie"     édite "amie" sur l'écran 25 x 80
```

Se reporter au paragraphe 1.2.5. du MANUEL DE PRESENTATION ET D'INSTALLATION DE GOUPIL pour toutes précisions complémentaires, notamment concernant les priorités des différentes cartes contrôleurs d'écrans.

Attention : Pour que PORT 4 fonctionne, il faut disposer d'un jeu de PROMs Moniteur GOUPIL récent. Toutes les configurations n'en sont pas forcément pourvues.

9. FICHIERS SEQUENTIELS

La forme simple d'utilisation des fichiers est la forme séquentielle. Par défaut, tous les fichiers de données sont créés sur le disque de Travail avec une extension .DAT (DATA en anglais)

9.1. OUVERTURE D'UN FICHIER

Avant d'utiliser un fichier sous SBASIC, il est nécesaire de l'ouvrir. L'ordre OPEN est utilisé pour cette opération. Il permet d'affecter un canal au fichier concerné.

Pour manipuler des fichiers séquentiels, il existe deux formes d'ordre OPEN :
- l'ouverture en vue d'une lecture, et
- l'ouverture en vue d'une écriture.

La syntaxe est la suivante :

OPEN OLD <expression chaîne> AS <num canal>
OPEN NEW <expression chaîne> AS <num canal>

Exemples :

```
OPEN OLD "TEST" AS 1
OPEN NEW A$ AS C%
```

L'ordre OPEN OLD ouvre le fichier spécifié dans <expression chaîne>, en lecture. <num canal> indique au BASIC, le numéro de canal qu'il doit utiliser pour ce fichier. Ce numéro de canal doit être compris entre 1 et 12, ce qui signifie qu'il existe une limite de 12 fichiers ouverts simultanément.

Si le fichier n'est pas trouvé, l'erreur 4 est signalée. Cette erreur peut être gérée par l'utilisation de "ON ERROR GOTO". Une fois ouvert, le fichier est prêt à être exploité. Le premier exemple ouvre, en lecture, le fichier "TEST.DAT" situé sur le disque de Travail en utilisant le canal 1.

L'ordre OPEN NEW est utilisé pour ouvrir un fichier en écriture, c'est-à-dire créer un nouveau fichier pour stocker des données. Le fichier spécifié dans l'expression chaîne ne doit pas déjà exister, sinon il sera détruit et un nouveau fichier portant le même nom sera créé sans aucun message d'avertissement. Le second exemple ouvre en écriture le fichier dont le nom est donné dans la variable A$. Le canal utilisé est donné par la valeur contenue dans C%.

9.2. ECRITURE DANS UN FICHIER SEQUENTIEL

Une version modifiée de l'ordre PRINT est utilisée pour écrire des données séquentielles sur un fichier disque. La syntaxe est la suivante:
<numligne> PRINT # <num canal>,<liste à écrire>

où <num canal> est le numéro de canal interne spécifié dans l'ordre OPEN correspondant. <liste à écrire> est la liste d'informations à écrire en utilisant les mêmes règles relatives à l'ordre PRINT.

Exemple :

```
10 OPEN NEW "ESSAI.DAT" AS 3
20 PRINT#3, "CECI EST UN FICHIER D'ESSAI"
30 PRINT#3, "CECI EST LA DEUXIEME LIGNE"
```

Ces lignes créent un nouveau fichier appelé "ESSAI.DAT" sur le disque de Travail. Le canal interne utilisé a la numéro 3. Ce numéro sera utilisé pour référencer ce fichier dans les autres instructions d'entrées-sorties.

Dans cet exemple, on crée deux enregistrements :
le premier est :
"CECI EST UN FICHIER D'ESSAI"

et le second est :
"CECI EST LA DEUXIEME LIGNE".

Voici un autre exemple permettant d'illustrer la création d'une table de nombres entiers de 1 à 5 et et de leur carré respectif sur disque :

```
100 OPEN NEW "CARRE" AS 1
110 FOR I=1 TO 5
120 PRINT#1, I, ",", I**2
130 NEXT I
```

Les fichiers créés dans ces exemples sont des fichiers de type "Texte" au sens FLEX-9, et peuvent donc être édités, listés, etc. avec n'importe quel utilitaire standard FLEX-9. L'édition du fichier CARRE.DAT, donne :

```
1 ,  1
2 ,  4
3 ,  9
4 , 16
5 , 25
```

9.3. LECTURE D'UN FICHIER SEQUENTIEL

Comme l'instruction PRINT, l'ordre INPUT sert à communiquer avec un fichier. La syntaxe de INPUT est :

<numligne> INPUT#<num canal>,<liste de données>
<numligne> INPUT LINE#<num canal>,<variable chaîne>

où <num canal> est le numéro de canal interne référencé dans l'ordre OPEN. La liste est la même que dans l'ordre INPUT.

L'ordre INPUT LINE permet de lire une ligne entière à partir du fichier et de la transférer dans la variable chaîne spécifiée :

Exemple :

```
10 OPEN OLD "NOMBRE" AS 2
20 INPUT#2, A, B
```

L'instruction 20 permet de lire les variables A et B sur le fichier disque NOMBRE.DAT. Le fichier fournit les données ASCII tout à fait comme si ces dernières étaient entrées au clavier. Les données lues doivent, bien entendu, correspondre en type et en format à celles qui y ont été écrites. Elles doivent être séparées par une virgule et se terminer par un retour chariot.

Exemple:

```
100 OPEN NEW "NOMBRE" AS 6
110 PRINT#6,A,",",B
120 CLOSE 6
130 OPEN OLD "NOMBRE" AS 6
140 INPUT#6, A, B
150 PRINT A, B:CLOSE 6
```

Ce programme crée sur disque, un fichier "NOMBRE.DAT", en écrivant les valeurs de A et B séparées par une virgule, ferme le fichier (voir plus loin l'ordre CLOSE), puis, l'ouvre en lecture, lit les valeurs de A et B et édite le résultat sur le terminal. On peut noter que la réouverture du fichier après fermeture repositionne le pointeur au début du fichier de telle façon que les données peuvent être lues à partir du début.

9.4. FERMETURE D'UN FICHIER

L'instruction CLOSE est utilisée pour refermer un fichier SBASIC. Cette fermeture libère le canal interne utilisé et recopie sur disque le ou les derniers enregistrements de son tampon. La syntaxe de la commande CLOSE est :

<numligne> CLOSE <num canal> (,<num canal>...)

La valeur de <num canal> est la même que celle utilisée dans l'ordre OPEN au moment de l'ouverture du fichier et indique le numéro de canal interne du fichier à fermer. On peut fermer plusieurs fichiers dans un seul ordre CLOSE.

Exemples :

```
200 CLOSE 3
520 CLOSE 1,6
```

Une erreur 43 n'est plus générée si le fichier n'a pas été ouvert auparavant.

9.5. INPUT# SUR LE CANAL 0

Le canal 0 est un canal un peu spécifique et il est réservé à la console (écran et clavier)

Il est parfois souhaitable d'entrer des données avec le terminal sans voir apparaître le "?" à chaque exécution de l'ordre INPUT ou INPUT LINE. Ceci est possible en spécifiant le numéro de canal 0.

Exemple :

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

Les INPUT de cet exemple se comportent exactement comme des instructions INPUT normales.

9.6. PRINT# SUR LE CANAL 0

Dans certains programmes, il est souvent nécessaire de prévoir un affichage (vers l'écran) et une impression (vers l'imprimante) après la réponse de l'utilisateur. Le canal 0 sert à rediriger les sorties vers l'écran ou vers l'imprimante selon l'état du canal 0.

Par défaut, il permet d'aiguiller les ordres PRINT en sortie, de la même façon qu'il permet d'aiguiller les ordre INPUT en entrée. L'utilisation du canal 0 sans ordre OPEN préalable permet l'affichage normal sur le terminal.

Exemple :

```
200 PRINT#0, "J'AFFICHE A L'ECRAN"
```

édite "J'AFFICHE A L'ECRAN" sur le terminal, comme si le "#0" n'avait pas été spécifié.

L'ouverture du canal 0 permet d'effectuer une sortie vers d'autres périphériques (une imprimante par exemple).

Le fait d'ouvrir un fichier avec OPEN sur le canal 0, ordonne au SBASIC son chargement en le considérant comme un fichier du type PRINT.SYS (pilote) et de l'utiliser comme interface de sortie chaque fois que PRINT#0 est rencontré.

Exemple :

```
10 OPEN "0.PRINT.SYS" AS 0
20 FOR I%=1 TO 10
30 PRINT#0,"J'ECRIS SUR L'IMPRIMANTE COURANTE"
40 NEXT I%
50 CLOSE 0
60 PRINT#0,"PUIS SUR L'ECRAN"
```

La ligne 10 demande a SBASIC de charger, à partir de l'unité 0 le fichier PRINT.SYS (SYS est l'extension par défaut quand le canal 0 est référencé). Notez qu'il n'est pas nécessaire de préciser OPEN OLD. Le format du fichier PRINT.SYS est décrit dans le manuel de programmation avancée de FLEX-9. Une fois que le fichier PRINT.SYS a été lu, toutes les sorties effectuées par l'ordre PRINT#0 transitent par ce programme système gérant l'imprimante. Si PRINT.SYS contient les programmes permettant la sortie parallèle toutes les éditions se feront sur l'imprimante connectée sur cette interface.

Les ordres PRINT ne contenant pas le #0 seront effectués sur l'écran. L'ordre CLOSE 0 indiquera à SBASIC d'aiguiller tous les PRINT#0 à nouveau sur l'écran, jusqu'à ce qu'un autre OPEN AS 0 soit rencontré.

L'exemple suivant illustre l'utilisation de PRINT sur le canal 0. Il permet le choix de l'imprimante ou de l'écran pour les éditions :

```
10 INPUT "SORTIES VERS ECRAN OU IMPRIMANTE (E ou I)";R$
20 IF R$ ="I" THEN OPEN "O.PRINT" AS 0
30 FOR I=1 TO 10
40 PRINT#0,I, SQR(I)
50 NEXT I
60 IF R$="I" THEN CLOSE 0
70 END
```

En mode direct, on peut faire des opérations sur les fichiers avec les instructions qui s'y rapportent, mais vous devez les refermer pour vous assurer que toutes les données sont bien recopiées et que le catalogue de la disquette est bien mis à jour.
Les instructions suivantes referment implicitement les fichiers :

- RUN
- CHAIN
- END
- CLOSE

10. LA GESTION DES PROGRAMMES IMPORTANTS

10.1 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 pendant l'exécution successivement plusieurs programmes. Les segments de programme doivent être soit de type .BAS, soit des programmes compilés de type .BAC. Chaque segment est un programme résidant sur disque pouvant être appelé par CHAIN. La syntaxe en est

<numligne> CHAIN <expression-chaîne> [<expression>]

Le programme référencé par <expression-chaîne> est chargé. Si le numéro du disque n'est pas indiqué, il le sera à partir du disque de Travail et aura l'extension .BAC (BAsic Compilé). La seconde <expression>, optionnelle, désigne le numéro de ligne où l'exécution du programme doit commencer. Notez l'absence de virgule entre les deux expressions.

Exemple :

```
1300 CHAIN "PROMENU.BAS" 1000
```

Cette ligne charge le fichier PROMENU.BAS et l'exécute à partir de la ligne 1000.

Chaîner un programme compilé (type .BAC), est plus efficace que chaîner un programme écrit en BASIC source (.BAS).
L'ordre CHAIN referme tous les fichiers ouverts, efface l'ensemble des variables et l'exécution du nouveau programme commence. Donc, tous les fichiers exploités et communs à plusieurs programmes doivent être ré-ouverts dans chacun d'eux et leurs variables ré-initialisées.

10.2 L'instruction OVERLAY

La mémoire utilisée par SBASIC est partagée en deux parties :
- la zone "Programme" et la zone "Données".

L'instruction OVERLAY permet de réserver une zone mémoire d'une certaine taille pour la partie "Programme", le reste étant alloué aux données. Ainsi, l'insertion de nouvelles lignes de programme n'affectera pas les données qui se trouvent donc protégées. Cette instruction est obligatoire si vous utilisez la possibilité d'Overlay, et elle soit se situer sur la première ligne du programme principal. Sa syntaxe est :

```
<numligne> OVERLAY <nbreligne>
```

où <nbreligne> est la taille mémoire réservée aux lignes du programme.
Cette taille doit être fournie en nombre de lignes qui doit être inférieur à 511. Une zone de 50 x <nbreligne> octets est alors réservée pour le programme, le reste de la mémoire étant affecté aux données.

Il sera ainsi possible de charger consécutivement un module (sous-programmes ou segment), de l'effacer quand celui-ci n'est plus utile et d'en appeler un autre etc... Une erreur 82 signale un débordement de mémoire lors d'un chargement d'un module programme. Il vous faut alors augmenter le paramètre donné après "OVERLAY".

Par exemple, le programme suivant :

```
  10 OVERLAY 50:GOTO 1000
 100 SUB UTIL1(A$) : INPUT A$ : RETURN
 200 SUB UTIL2(A$) : PRINT A$ : RETURN
1000 DELETE 2000,3000: REM Zone où l'on charge les fonctions
1005 PRINT "Que voulez vous faire "
1008 PRINT "0:Arreter 1:Modifier A$ 2:Editer A$ ?"
1100 R$=INCH$(0) :IF INSTR(1,"012",R$)=0 THEN 1100
1120 IF R$="0" THEN END
1200 FF$ = "fonct"+R$
1205 LOAD FF$,,2500:EXECUTE "CALL FF$" + "(A$)":GOTO 1000
```

Ce programme réserve une zone de 50 lignes pour y placer des lignes de programme. On trouve alors dans cette zone (en 100 et en 200), les sous-programmes utilitaires SUB UTIL1 et SUB UTIL2. Ces derniers seront utilisés par chacun des sous-programmes fonctionnels chargés par la suite entre 2000 et 3000. La ligne 1000 efface les lignes comprises entre 2000 et 3000 avant d'y charger l'un des deux sous-programmes fonctionnels suivants dont le nom se trouve dans la variable FF$.

- Le premier sous-programme devant figurer sur le disque s'appelle FONCT1.BAS.

```
10 SUB FONCT1(A$)
20 CALL UTIL1(A$)
30 CALL UTIL2("*******"+A$)
40 RETURN
```

Il sera chargé sur les lignes 2510, 2520, 2530, 2540 grâce au déplacement de 2500 que l'on trouve en ligne 1205.

- Le second sous-programme qui doit se trouver sur le disque s'appelle FONCT2.BAS.

```
 5 SUB FONCT2(A$)
15 CALL UTIL2("<<<<<"+A$)
25 RETURN
```

Il sera chargé sur les lignes 2505, 2515, 2525.

En fonction du choix que fait l'utilisateur en ligne 1100, la ligne 1200 détermine la variable FF$. La ligne 1205 permet de charger puis d'appeler les SUBroutines fonctionnelles FONCT1 et FONCT2, et de revenir en 1000.

Lorsque le programme s'arrête, seules les lignes 10 à 1205 restent en mémoire centrale, puisque la ligne 1000 détruit les lignes 2000 à 3000. Toutefois, un STOP convenablement placé en début de ligne 1000 permettrait de pouvoir visionner les lignes 2000 à 3000.

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

11. LA GESTION DES ERREURS

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

- Les erreurs d'Entrée-Sortie sur disque ou sur le terminal écran.
- Les erreurs de syntaxe ou d'exécution.

La liste complète des numéros d'erreur et leur signification est donnée dans l'appendice A. Les numéros d'erreur sont compris entre 1 et 49 pour les erreurs d'E-S, les numéros supérieurs à 49 étant relatifs à la deuxième catégorie. On peut noter que les erreurs 1 à 28 sont des erreurs d'E-S sur disque et sont les mêmes que celles signalées par FLEX-9. En général SBASIC édite l'erreur et arrête l'exécution du programme. Il est parfois utile de pouvoir continuer l'exécution du programme notamment lorsque les erreurs sont du premier type avec l'ordre ON ERROR.

11.1. L'ORDRE ON ERROR GOTO

L'ordre ON ERROR GOTO est utilisé pour indiquer à SBASIC à quel numéro de ligne il doit se brancher en cas d'erreur. Chaque fois qu'une erreur est détectée, SBASIC vérifie si un ordre ON ERROR GOTO a été exécuté. Si tel est le cas, le contrôle est transféré à la ligne indiquée. Si aucune ligne n'est indiquée, le programme s'arrête et édite un message d'erreur de manière usuelle.

La syntaxe en est :

<numligne> ON ERROR GOTO (<numligne>)

Pour être efficace cette instruction doit être exécutée avant toute instruction susceptible de provoquer une erreur. En cas d'erreur, les variables réservées système ERR et ERL renferment respectivement le numéro d'erreur et le numéro de ligne où l'erreur a été enregistrée (pour plus d'informations sur ces variables, voir 11.3).

11.2. L'INSTRUCTION RESUME

L'instruction RESUME est utilisée pour revenir au programme principal dès que la sous-routine d'erreur a été exécutée.

Si pendant l'exécution du programme, une erreur d'E-S est signalée et qu'une instruction ON ERROR GOTO a été exécutée auparavant, SBASIC passera le contrôle à la première ligne de la routine d'erreur. L'instruction RESUME doit être utilisée obligatoirement pour redonner le contrôle au programme principal et sortir de la routine d'erreur sinon une erreur 66 sera générée.

La syntaxe en est.:

<numligne> RESUME (<numligne>)

Si un numéro de ligne est fourni après RESUME, SBASIC reprendra l'exécution du programme à partir du début de cette ligne. Si aucun numéro de ligne n'est fourni (ou est égal à 0) SBASIC reprendra l'exécution à partir du début de la ligne où l'erreur s'est produite :

Exemples :

```
1000  RESUME

2000  RESUME 200
```

Linstruction du premier exemple fait reprend l'exécution du programme à la ligne qui a causé l'erreur. La ligne 2000 permet de reprendre l'exécution à partir de la ligne 200.

11.3. ERR et ERL : VARIABLES SYSTEMES

En cas d'erreur, les deux variables ERR et ERL renferment :
- ERR : le numéro de l'erreur ;
- ERL : le numéro de la ligne où l'erreur s'est produite.

Ces variables système ne peuvent pas être affectées par l'utilisateur mais sont disponibles en lecture seule à tout moment.

Exemple :

```
32000 IF ERR=4 THEN PRINT "FICHIER NON TROUVE"
32020 IF ERR=8 AND ERL=1500 THEN PRINT "FIN DE FICHIER"
```

La ligne 32000 provoque l'édition du message "FICHIER NON TROUVE" si le dernier numéro d'erreur est égal à 4.
La ligne 32020 affichera "FIN DE FICHIER" si une erreur 8 s'est produite en ligne 1500.

11.4. UTILISATION DE ON ERROR GOTO

Parfois, seules certaines sections de programmes nécessitent l'utilisation de ON ERROR. Il est possible d'annuler l'effet d'un ON ERROR GOTO qui n'est plus utile de la manière suivante :

<numligne> ON ERROR GOTO

<numligne> ON ERROR GOTO 0

Ces deux expressions sont équivalentes et annulent tous les anciens ordres ON ERROR GOTO. De plus, en cas d'erreur, ON ERROR GOTO 0 arrêtera le programme en éditant un message d'erreur.

Il est à noter que cette instruction est sans effet si l'on est en cours de traitement d'une erreur dans la routine d'erreur et si l'on a pas exécuté une instruction RESUME.

11.5. EXEMPLES DE GESTION D'ERREURS

Entrée de nombres au clavier :

```
10 ON ERROR GOTO 1000
20 INPUT "ENTREZ 3 NOMBRES", A, B, C
30 PRINT "LA SOMME EST "; A+B+C
40 GOTO 20
1000 IF ERR<>30 THEN GOTO 1030
1010 PRINT "NE TAPEZ SEULEMENT QUE DES NOMBRES - recommencez"
1020 RESUME
1030 IF ERR=108 THEN ?"NOMBRE TROP GRAND - recommencez":RESUME
1040 PRINT "ERREUR";ERR;"A LA LIGNE";ERL
1050 END
```

Lecture et affichage d'un fichier texte :

```
10 ON ERROR GOTO 1000
20 PRINT "TAPEZ LE NOM D'UN FICHIER TEXTE AVEC SON EXTENSION";
25 INPUT LINE F$:IF F$="" THEN PRINT "FIN" : END
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":RESUME 100
1010  IF  (ERR=4)  AND  ((ERL=30) OR (ERL=40)) THEN PRINT  "Fichier
     inconnu !":RESUME 100
1100 ON ERROR GOTO
```

12. LES FICHIERS TABLEAUX VIRTUELS

Les tableaux virtuels constituent une classe particulière de fichiers disque. Il existe, en SBASIC deux types de fichiers à accès direct : le premier que nous allons décrire dans ce paragraphe est le fichier à accès direct par éléments ou tableau virtuel ; le second, que nous verrons plus loin, est le fichier à accès direct par enregistrement.

Le tableau virtuel est l'exemple le plus simple de fichier à accès direct. C'est un moyen de stocker sur disque un ensemble de données de la même manière qu'en mémoire centrale, c'est-à-dire en les référençant par un nom de tableau. Les avantages d'un tableau virtuel sont :
qu'il peut avoir une taille quelconque, donc une dimension bien plus grande que s'il résidait en mémoire centrale, que les données sont stockées sur disque dans un fichier, que les éléments du tableau sont accessibles en permanence comme en mémoire centrale en lecture comme en écriture quelque soit son emplacement sur disque, ce qui autorise puissance et simplicité d'emploi.

REMARQUE IMPORTANTE : Un fichier à accès direct (classe dont fait partie les tableaux virtuels) ne peut être lu en mode séquentiel avec les instructions PRINT# et INPUT#.

12.1. OUVERTURE D'UN FICHIER A ACCES DIRECT

Avant d'être référencé, un fichier doit être ouvert. Nous allons voir dans ce paragraphe comment ouvrir un fichier à accès direct, qu'il soit tableau virtuel ou à accès par enregistrement. L'instruction OPEN revêt trois formes pour ouvrir un fichier à accès direct :

```
<numligne> OPEN OLD <nom fichier> AS <num canal>
<numligne> OPEN NEW <nom fichier> AS <num canal>
<numligne> OPEN <nom  fichier> AS <num canal>
```

OPEN OLD ouvre un fichier qui doit exister préalablement sur disque. S'il n'est pas trouvé, FLEX-9 signale l'erreur 4.

OPEN NEW crée un nouveau fichier. S'il existe un fichier ayant les mêmes spécifications (lecteur.nom fichier.extension), celui-ci est détruit.

OPEN permet soit d'ouvrir un fichier existant, soit d'en créer un nouveau si <nom fichier> n'existe pas.

L'ouverture du fichier n'est effective qu'au moment de la première entrée-sortie. C'est pour cette raison que les erreurs se produisant à l'ouverture ne sont pas signalée à l'exécution de l'ordre OPEN mais au moment de la première lecture ou de la première écriture.

12.2. DECLARATION D'UN TABLEAU VIRTUEL

Un tableau virtuel doit être déclaré par une forme spéciale de l'instruction DIM :

<numligne> DIM # <num canal>,<variable> (=<dimension>)

où <num canal> désigne un numéro de canal interne compris entre 1 et 12, canal sur lequel le fichier a été ouvert.
<variable> est le tableau dimensionné qui sera associée au canal d'entrée-sortie indiqué.
<dimension> n'est utilisé que pour les tableaux de chaînes.
Le tableau peut être en simple ou double dimension (liste ou matrice).
Exemple :

```
20 DIM#3, A(100,50)
30 DIM#4, B%(100)
40 DIM#9, ECRIT$(5000)=42
```

La première déclaration définit la matrice A de dimension 101 par 51, associée au canal 3 (la base du tableau commence à 0) ; la seconde définit une liste de valeurs entières de 101 éléments associée au canal 4. Les données sont rangées dans le fichier suivant un format interne, c'est-à-dire 2 octets pour un entier, 8 octets pour un réel, et sous la forme d'une suite de code ASCII pour une chaîne de caractères.

La manipulation des tableaux virtuels ou des tableaux standards en mémoire se fait de la même manière, à trois exceptions près :

- alors qu'en mémoire les chaînes d'un tableau sont de longueur quelconque, sur disque les chaînes d'un tableau virtuel sont nécessairement toutes de même longueur.
- la taille maximum d'une chaîne d'un tableau virtuel est limitée par la taille du secteur disque, soit 252 caractères, et ne peut donc être définie qu'entre 1 et 252 inclus. Les chaînes seront tronquées à droite si elles dépassent cette longueur maximum.
- on ne peut permuter le contenu de deux éléments d'un tableau virtuel avec l'instruction SWAP, ni utiliser l'instruction SET.

La définition d'un tableau virtuel de chaînes de caractères, s'effectue de la manière suivante :

<numligne> DIM # <num canal>,<chaîne (<dimension>)=<expression>>

ou <expression> est la longueur commune des éléments du tableau .

Par exemple:

```
100 DIM#7,CLIENT$(100)=63
```

L'<expression> à droite du signe égal précise la longueur en nombre de caractères de chaque chaîne. L'exemple ci-dessus déclare un tableau CLIENT$ de 101 rangées de 63 caractères. Si la longueur n'est pas précisée dans la déclaration, elle sera de 18 caractères par défaut.

Il faut préciser que pour obtenir le rangement efficace sur disque d'un tableau virtuel de chaînes de caractères, il est nécessaire de tenir compte de la taille d'un secteur (252 octets). Un secteur peut contenir une ou plusieurs chaînes, mais une chaîne doit être entièrement contenue dans un secteur. D'où la limite de 252 caractères et l'intérêt de prendre comme longueur un diviseur de 252 pour ne pas perdre d'espace disque.

Exemple :

```
10 DIM#1, DOSSIER$(100)
```
la longueur par défaut étant 18 , il y a 14 chaînes par secteur et pas de perte d'espace puisque 14 x 18 = 252 .

```
20 DIM#2, B$(100)=63
```
il y a 4 éléments par secteur : pas d'espace perdu.

```
30 DIM # 3,C$(20)=130
```
dans ce dernier cas, on perd 122 octets par secteur : 252 - 130 = 122, on ne peut mettre qu'une chaîne par secteur.

12.3. UTILISATION DES TABLEAUX VIRTUELS

Après l'ouverture du fichier et la déclaration du tableau virtuel avec l'instruction DIM, on peut procéder à des affectations ou lire les éléments comme on le fait avec un tableau standard, mais, avec les restrictions exposées précédemment

Un exemple en montre l'utilisation :

```
10 OPEN "TESTFILE" AS 1
20 DIM#1,A(100)
25 A(100)=0 : REM Ecriture du dernier élément pour initialiser
              le fichier la première fois.
30 INPUT "DONNER INDICE, NOUVELLE VALEUR" ; IND,NVAL
40 PRINT "VALEUR COURANTE" ; A(IND)
50 A(IND)=NVAL
60 PRINT "NOUVELLE VALEUR :";A(IND)
70 CLOSE 1
80 END
```

Ce programme ouvre le fichier TESTFILE.DAT sur l'unité de travail. (unité 1). S'il n'existe pas, un nouveau fichier est créé (3ème forme d'OPEN).
La ligne 20 déclare un tableau virtuel A de réels contenant 101 éléments.
La ligne 25 permet de créer le dernier élément du tableau de manière à ce que tous ceux le précédent soient accessibles et initialisés à 0, en effet, SBASIC allouera sur disque, dans ce cas, la place nécessaire au stockage du fichier intégral.
Le programme traite ensuite le tableau A de façon standard : impression de l'élément A(IND), modification de sa valeur et impression de la nouvelle valeur.
La ligne 70 ferme le fichier qui existe désormais sur le disque, avec la valeur de l'élément A(IND). La fermeture du fichier est obligatoire, car elle permet la recopie effective de son tampon d'entrée-sortie.

12.4. REMARQUES SUR LES TABLEAUX VIRTUELS

Un fichier créé et référencé comme tableau virtuel ne contient pas d'informations sur la dimension du tableau et le type de données contenues. Un secteur peut contenir 31 réels ou 126 entiers ou un nombre de chaînes en fonction de leur longueur.

Une matrice est rangée par lignes, c'est-à-dire la ligne 0 au début du fichier, suivie de la ligne 1, etc.
Ce qui nous fait remarquer que :

L'accès séquentiel aux éléments d'une matrice est plus rapide par ligne que 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 du secteur.

Cette remarque s'applique aussi, lorsque l'on effectue un décalage, ou la permutation de deux éléments. Ne pas en tenir compte peut mener à un nombre énorme d'accès disque, et allourdir considérablement un traitement.

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

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

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

Le fichier disque correspondant contient alors 2500 éléments (nombres flottants) 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 déclaré ultérieurement en lecture par une autre instruction comme :

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

(qui permet l'accès aux 1001 premiers éléments seulement).

12.5. EXTENSION D'UN TABLEAU VIRTUEL

Lors de sa création, un fichier à accès direct ne contient qu'un seul secteur d'initialisé (à 0 pour les nombres et à des chaînes non significatives pour les éléments de type caractères). La référence en lecture à un élément d'un tableau virtuel extérieur à ce secteur (c'est-à-dire, qui n'a jamais été écrit) provoquera l'erreur 24 semblable à une erreur "enregistrement non existant" ou "fin de fichier". Un fichier à accès direct peut être étendu à tout moment en affectant une valeur non significative à un élément du tableau virtuel extérieur au fichier. Chaque fois qu'une extension est demandée, SBASIC ajoute au fichier quelques secteurs de plus que nécessaire, anticipant ainsi une future extension. L'extension d'un fichier peut prendre un temps non négligeable, proportionnel au nombre de secteurs à rajouter au fichier.
Ce principe est identique pour les fichiers à accès direct par enregistrement.

Exemple :

```
10 OPEN NEW "AFFAIRES.DAT" AS 1
20 DIM#1, A(250)
30 PRINT A(250)
```

Provoquera l'erreur 24, car l'élément 250 n'existe pas dans le fichier nouvellement créé. Pour contenir 250 éléments le fichier doit être initialisé (ou étendu) de la façon suivante :

```
10 OPEN NEW "AFFAIRES.DAT" AS 1
20 DIM#1, A(251)
25 A(251)=0
30 PRINT A(250)
```

Lors d'une extension, comme dans cet exemple, les données précédent l'indice 251 ont une valeur nulle (les chaînes de caractères n'ont jamais une valeur dite "nulle" dans un fichier mais peuvent être initialisée comme des chaînes d'espaces). Dans le fichier créé, les éléments A(0) à A(250) existent maintenant et valent tous zéro.

13. FICHIERS A ACCES DIRECT PAR ENREGISTREMENT

Nous avons décrit jusqu'à présent deux méthodes d'entrée-sortie disque, la méthode séquentielle avec les instructions PRINT# et INPUT# , et à 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 un troisième type de fichier-disque : l'accès direct par enregistrement. Les données sont directement accessibles à l'aide d'un numéro d'enregistrement relatif au début du fichier et tous les types de données (entier, réel et chaîne) peuvent être stocké dans un enregistrement.

Cette méthode offre la possibilité de stocker des données qui seront rangées sur disque dans des enregistrements physiques (les secteurs) de longueur donnée, 252 octets. Chaque enregistrement ou secteur d'un fichier peut être lu, écrit, ou ré-écrit sur demande, quelque soient les données qu'il contient. Une chaîne de caractères peut être placée à côté d'une donnée numérique binaire, entière ou réelle.
La méthode est donc efficace et rapide pour traiter des enregistrements composés de sous-ensembles de données de type différent.

13.1. OUVERTURE ET FERMETURE DES FICHIERS A ACCES PAR ENREGISTREMENT

Ces fichiers sont semblables à ceux utilisés pour stocker les tableaux virtuels. Ils sont ouverts et fermés par les instructions OPEN et CLOSE (voir paragraphe 12.1). Il faut noter que toute tentative d'accès direct sur un fichier séquentiel provoquera une erreur.

13.2. COMMUNICATION AVEC LE DISQUE

13.21 PRINCIPES DE BASE :

Ce type de fichier communique avec la mémoire centrale de l'ordinateur par un tampon de 252 caractères qui contiendra l'enregistrement en cours de traitement. L'instruction FIELD# , permet de définir les zones de ce tampon, chaque zone étant accessible par une variable de type chaîne alphanumérique associée.

Les instructions LSET, RSET ou SET autorisent la modification des variables descriptives de ce tampon.

Les instruction GET et PUT (que nous verrons dans le chapitre 13.3), permettent respectivement le transfert des données d'un enregistrement du fichier vers le tampon et vice-versa. Il faut donc au préalable définir les différentes zones de ce tampon ainsi que les variables associées.

13.22 L'instruction FIELD # :

Sa syntaxe est la suivante :

<numligne> FIELD # <num canal>,<lg1> AS <chaîne1>,<lg2> AS <chaîne2> ...)

où <num canal> désigne le canal interne utilisé lors de l'ouverture du fichier avec l'instruction OPEN. Plusieurs zones peuvent être ainsi définies dans la limite des 252 octets du tampon.

A l'ouverture du fichier, un tampon est créé. FIELD # suivit de la liste des variables, est le moyen de communication entre la mémoire et le tampon disque.

Exemple :

```
10 OPEN "SALARIE.DAT" AS 1
20 FIELD#1, 20 AS NOM$, 10 AS PRE$,15 AS SERV$,20 AS QUAL$
30 FIELD#1, 30 AS NOMETPRENOM$,35 AS EMPLOI$
```

Nous avons ainsi découpé le tampon d'entrée-sortie associé au canal 1, de 2 façons : d'une part :

première zone de 20 caractères pour le nom NOM$,
seconde zone de 10 caractères pour le prénom PRE$,
troisième zone de 15 caractères pour le service SERV$,
quatrième zone de 20 caractères pour la qualification QUAL$.
le reste n'étant pas défini.

et d'autre part :

première zone de 30 caractères pour le Nom et Prénom,
seconde zone de 35 caractères pour l'Emploi.
le reste n'étant pas défini.

Si le programme modifie par exemple la variable NOM$, la variable NOMETPRENOM$ est alors modifiée automatiquement puisque ces variables sont associées à la même zone du tampon.

Il faut noter l'aspect dynamique d'une telle déclaration de variable par FIELD. Ainsi lorsque l'on exécute dans la suite du programme la ligne :

```
200 FIELD # 1, 65 AS SALARIE$
```

on aura un nouveau descripteur du tampon d'entrée-sortie. Il suffira alors de ne plus utiliser les variables NOM$, PRE$, SERV$, QUAL$ précédentes si l'on ne désire faire qu'un traitement de bloc sur SALARIE$.

13.23 LSET et RSET :

Ces deux instructions (ainsi que SET, expliquée dans le chapitre 6.7) autorisent la modification des données contenues dans le tampon du fichier. Attention ! elles ne modifient pas le contenu du fichier stocké sur disque tant que l'instruction "PUT #" n'a pas été utilisée (voir plus loin).

Toute affectation d'une variable associée au tampon avec un LET implicite ou explicite, ne modifie pas la valeur de la variable tampon, mais une autre variable est créée annulant son association avec le tampon.

13.24 EXEMPLES D'EXPLOITATION SUR UN TAMPON DE FICHIER :

Voici quelques exemples mettant en évidence les propos exposés ci-avant :

Supposons qu'un enregistrement de Matériel contienne 14 zones de 18 caractères (14 x 18 = 252), zone sudivisée en 2 rubriques de 10 et 8 caractères respectivement. Pour accéder à la cinquième zone de l'enregistrement, on peut écrire l'instruction FIELD # suivante :

```
120 FIELD#1,(5-1)*18 AS MU$, 10 AS COMP$, 8 AS FABR$
```

La variable MU$ (dite "muette") sert ici à positionner les deux rubriques COMP$ (composant) et FABR$ (fabricant) dans le tampon, en réservant les 72 premiers (4 x 18) caractères.

Une forme plus générale permet de définir dynamiquement chacune des 14 zones (ce qui serait fastidieux avec l'instruction FIELD décrite précédemment) :

```
50 DIM COMP$(13),FABR$(13)
100 FOR I%=0 TO 13
150 FIELD#1, I%*18 AS MU$, 10 AS COMP$(I%), 8 AS FABR$(I%)
200 NEXT I%
```

Nous avons associé les tableaux des Composants (COMP$(13)) et des Fabricants (FABR$(13)) du Matériel au tampon d'entrée-sortie du fichier ouvert sur le canal numéro 1.

13.3. LES INSTRUCTIONS GET ET PUT

Ces instructions ordonnent le transfert des données contenues dans le tampon du fichier vers le support disque par bloc de 252 octets ou inversement.

Leur syntaxe est la suivante:

<numligne> GET # <num canal> (, RECORD <expression>)
<numligne> PUT # <num canal> (, RECORD <expression>)

- <num canal> représente le numéro du canal interne (de 1 à 12) sur lequel le fichier a été ouvert.

- La partie RECORD est optionnelle : si elle n'est pas spécifiée , l'enregistrement suivant est lu ou écrit (accès séquentiel). Si elle est présente, l'expression qui suit le mot RECORD donne le numéro de l'enregistrement à lire ou à écrire. Le premier enregistrement porte le numéro 1.

L'instruction GET ordonne la lecture et le transfert de l'enregistrement précisé vers le tampon d'entrée-sortie associé au canal utilisé. Toutes les variables définies précédemment dans l'instruction FIELD# sont alors modifiées et immédiatement accessibles.

Inversement l'instruction PUT écrit sur disque le contenu du tampon dans l'enregistrement spécifié, sans modifier la valeur des variables. Comme les systèmes d'exploitation ont tendance a optimiser les accès disque, il est probable que le tampon ne soit pas immédiatement et physiquement recopié sur disque.

Les enregistrements d'un fichier sont numérotés de 1 à n, n étant la taille du fichier (soit : le plus grand numéro d'enregistrement écrit). Tenter de lire un enregistrement de numéro supérieur à n provoquera une erreur 24 (ou 8 si l'accès séquentiel a été utilisé). Ecrire un enregistrement de numéro supérieur à n étendra automatiquement le fichier.

Tout accès séquentiel (sans préciser RECORD) postérieur à un accès sélectif (ou précisant RECORD) se fera sur l'enregistrement qui suit celui sur lequel on était positionné (c'est à dire celui correspondant à l'accès sélectif).

Par exemple :

```
10 OPEN "BIBLIO.DAT" AS 2
20 FIELD#2, 252 AS BOUQUIN$
30 GET#2, RECORD 25
40 PUT#2
50 GET#2
```

La ligne 30 permet de lire l'enregistrement numéro 25, la ligne 40 provoque l'écriture du tampon dans l'enregistrement 26, la ligne 50 lit l'enregistrement 27. S'il n'existe pas alors erreur 24.

13.4. EXTENSION DES FICHIERS ACCEDES PAR ENREGISTREMENT

La procédure de création et d'extension est la même que pour les fichiers de tableaux virtuels. A la création, le fichier ne contient qu'un enregistrement (un secteur). Tenter de lire un enregistrement inexistant (jamais écrit) provoquera une erreur 24. Le fichier peut être étendu à tout moment par l'écriture d'un enregistrement dont le numéro est supérieur à celui du dernier enregistrement écrit.

Chaque fois qu'une extension est demandée, SBASIC ajoute parfois quelques secteurs de plus que nécessaire au fichier, anticipant une future extension. L'extension d'un fichier peut prendre un temps non négligeable selon le nombre de secteurs à lui rajouter. C'est pourquoi nous vous recommandons d'initialiser vos fichiers à une taille nominale. Vous gagnerez du temps d'exploitation.

Par exemple : Soit un fichier dont on suppose une taille nominale de 500 enregistrements ;

```
10 OPEN NEW "ANNEE.DAT" AS 1
20 FIELD#1, 120 AS MOIS$
30 GET#1, RECORD 20
```

Ce programme provoque une erreur 24, le fichier étant créé avec un seul secteur. Il est donc nécessaire d'étendre le fichier avant toute lecture de l'enregistrement numéro 20 :

```
== ce qu'il faut faire ==

10 OPEN NEW "ANNEE.DAT" AS 1
20 FIELD#1, 120 AS MOIS$
30 PUT#1, RECORD 500

40 GET#1, RECORD 20
...etc...
```

```
== ne pas faire ==

10 OPEN NEW "ANNEE.DAT" AS 1
20 FIELD#1, 120 AS MOIS$
30 FOR I%=1 TO 500:PUT#1,
   RECORD I%
40 NEXT I%:GET#1,RECORD 20
...etc...
```

Tous les nouveaux enregistrements créés lors d'une extension sont mis à zéro (caractères ASCII nul), à l'exception de l'enregistrement référencé dans l'instruction PUT, qui peut contenir soit les données situées dans le tampon, soit des espaces (notamment pour les chaînes de caractères).

13.5.EXEMPLES

Exemple 1 :

```
10 OPEN "CARTES.DAT" AS 1
20 FIELD#1, 15 AS ROI$, 20 AS DAME$, 30 AS FOU$
25 INPUT "No ENREG"; NE%:IF NE%=0 THEN 110
30 GET#1, RECORD NE%
40 PRINT "*"; ROI$; "*"
50 PRINT "*";DAME$; "*"
60 PRINT "*"; FOU$; "*"
70 LSET ROI$  = "1 ROI DE PIQUE"
80 LSET DAME$ = "1 DAME DE COEUR"
90 RSET FOU$  = "NOUVEAU FOU JUSTIFIE A DROITE"
100 PUT # 1,RECORD NE%
105 GOTO 25
110 CLOSE 1
120 END
```

Le fichier CARTES.DAT est ouvert sur le disque de travail, et trois zones ROI$, DAME$ et FOU$ sont définies dans l'enregistrement correspondant. L'enregistrement spécifié est lu (il faut qu'il existe !) et l'on affiche les trois zones. Les lignes 70, 80 et 90 modifient le tampon qui est ensuite recopié dans le fichier par la ligne 100. L'omission de cette ligne dans le programme aurait laissé le fichier inchangé.

Exemple 2

Ce programme range dans un enregistrement les éléments d'un tableau de réels :

```
10 DIM A(30), A$(30)
20 OPEN "REELS.DAT" AS 1
30 FOR I%=0 TO 30:REM Init. Field tampon
40 FIELD#1, 8*I% AS MUETTE$, 8 AS A$(I%)
50 NEXT I%
200 FOR I%=0 TO 30:REM Ranger nombres vers tampon
210 LSET A$(I%)=CVTF$(A(I%))
220 NEXT I%
230 PUT#1
240 CLOSE 1
250 END
```

Le fichier FPDATA est ouvert après la déclaration des tableaux A et A$, et les zones sont définies dans l'enregistrement de façon à réserver 8 octets par élément du tableau de réels. Entre les lignes 30 et 200 des valeurs peuvent être affectées aux éléments de A, qui ensuite sont transférées dans le tampon au moyen de LSET et de la fonction de transfert de binaires CVTF$. L'enregistrement est écrit dans le fichier à la ligne 230 (1er enregistrement), et le fichier est finalement fermé.

Exemple 3

Considérons un fichier Salariés où chaque enregistrement contient certains renseignements sur un employé et où le numéro d'enregistrement l'identifie dans le fichier. Le programme imprime le numéro de téléphone du salarié dont on a donné le numéro d'identification et propose sa modification.

Le nom de l'employé est rangé dans les 20 premiers caractères de l'enregistrement et le numéro de téléphone dans les positions 90 à 104.

```
10 OPEN "EMPLOYE" AS 1
20 FIELD#1, 20 AS NOM$, 69 AS DIV$, 15 AS TEL$
25 PUT#1, RECORD 101:REM Initialisation Fichier

30 INPUT "NUMERO DE L'EMPLOYE"; E%
40 IF E%<=0 THEN 200 ELSE IF E%>100 THEN PRINT CHR$(7):GOTO 30
50 GET#1, RECORD E%
60 IF NOM$<"0" THEN GOTO CREAT
70 PRINT "NOM =";NOM$ : PRINT "TEL =";TEL$
75 PRINT "MODIFICATION DU No DE TELEPHONE (O/N) "
80 R$=INCH$(0):PRINT R$
85 IF (R$<>"O") AND (R$<>"o") THEN 30

90 PRINT "NOUVEAU No";: INPUT LINE NTEL$
100 LSET TEL$=NTEL$
110 PUT#1, RECORD E%
120 PRINT "Enregistr.modifié"
130 GOTO 30

150 LABEL CREAT
160 INPUT "Nom ";N$
170 PRINT "Tel ";: INPUT LINE NTEL$
180 LSET NOM$=N$ : GOTO 100

200 CLOSE 1:END
```

Le fichier EMPLOYE est ouvert sur le canal 1, la variable DIV$ est utilisée pour avoir un positionnement sur le numéro de téléphone dans l'enregistrement, NOM$ contenant le nom de l'employé. La ligne 30 demande le numéro identifiant l'employé, pour lire l'enregistrement. La ligne 40 vérifie que ce numéro est inférieur à 100. S'il est négatif ou nul, le programme referme le fichier puis s'arrête.

En ligne 60, si le Nom de l'employé appelé est non significatif (<"0") alors, on demande les éléments permettant la creation de son dossier.

Si le numéro de téléphone de l'employé est modifié, l'enregistrement est réécrit dans le fichier en ligne 110.

14. LA FONCTION USR

La fonction USR permet d'exécuter sous SBASIC un sous-programme écrit en langage machine. Une valeur codée sur 16 bits peut être passée au sous-programme comme argument de la fonction USR, et une valeur codée également sur 16 bits peut être retournée au programme SBASIC comme valeur de la dite fonction.

Syntaxe :

<numligne> <var num1> = USR(<var num2>)

Les explications qui suivent sont propres aux Système FLEX-9.

Il faut d'abord réserver un espace mémoire nécessaire au stockage du sous-programme :

- soit en utilisant la zone de chargement des utilitaires du Flex, c'est-à-dire de $C100 à $C6FF ;
- soit en utilisant la partie de mémoire laissée libre par SBASIC en $EC00 à $EFFF, à condition de ne pas exploiter des programmes s'y chargeant aussi ;
- soit en modifiant le pointeur de fin de mémoire ($CC2B/$CC2C de FLEX-9) de façon à disposer d'un emplacement où charger le sous-programme, c'est-à-dire juste au-dessous de $BFFF.

Quand SBASIC rencontre une fonction USR dans une expression, il évalue l'argument <var num2>, le convertit en un entier binaire (16 bits, arithmétique en complément à 2) et le place dans les octets d'adresse $26 et $27. Ensuite SBASIC recherche dans les deux octets d'adresse $24 et $25 l'adresse d'exécution du sous-programme. Si cette adresse est nulle une erreur 90 ou 91 est signalée. Dans le cas contraire, SBASIC exécute un JSR à l'adresse spécifiée. L'adresse peut être placée manuellement en $24 et $25 au moyen de la fonction DPOKE.

Au niveau du sous-programme, il faut accéder aux octets $26 et $27 pour trouver l'argument fourni par SBASIC. En retour, on peut transmettre un résultat en le plaçant dans ces mêmes octets. Il sera retourné dans <var num1> du programme SBASIC. Si le pointeur de pile a été modifié, il faut le restaurer. Le sous-programme doit se terminer par un RTS. Celui-ci redonne le contrôle au BASIC. Le sous-programme ne doit pas utiliser plus de 256 octets de la pile pendant son exécution.

Considérons un programme en assembleur permettant de transférer directement en mémoire le contenu d'un secteur disque donné en piste/secteur P% et S% (attention au danger !).

Voici comment procéder pour mettre en oeuvre cette fonction USR :

```
PROGRAMME ASSEMBLEUR :                    CODE 6809
     LDX     #$C980 *Pointeur FCB         8E C980
     CLRA                                 4F
     STA     $26    *No piste             97 26
     STA     $27    *No secteur           97 27
     JSR     >$D406 *Appel FMS            BD D406
     BNE     SUITE                        26 01
     RTS                                  39
   SUITE
     LDA     #$7                          86 07
     JSR     PUTCHR *Err:Bip              BD CD18
     LDA     $C981  *Erreur rendue        B6 C981
     STA     $27    *qu'on stocke         97 27
     RTS                                  39
     END
```

Dans cet exemple (réservé aux personnes ayant une bonne connaissance du FMS FLEX), on considère que l'image du secteur sera contenue dans le tampon du FCB d'adresse $C980 et l'on placera le programme assembleur en $C120. Avant de faire ce qui suit, on recommande à l'utilisateur de faire une sauvegarde de ses disques, car la moindre erreur peut être fatale...
Ce programme est donné en exemple, mais aux risques et périls des disques traités !

DANS LE PROGRAMME SBASIC:

Commençons par placer le programme assembleur :

```
100 RESTORE 200
120 FOR I=0 TO 24:READ H$:POKE I+HEX("C120"),HEX(H$):NEXT I
200 DATA 8E,C9,80,4F,97,26,97,27,BD,D4,06,26,01,39
220 DATA 86,07,BD,CD,18,B6,C9,81,97,27,39

250 INPUT "No PISTE";P%:INPUT "No SECTEUR";S%
260   POKE   HEX("C99E"),P%:POKE   HEX("C99F"),S% REM   saisie
Piste/Sect.
280 POKE HEX("C983"),0:REM No Lecteur Disque
300 DPOKE HEX("24"),HEX("C120"):REM Adresse debut programme
320 POKE HEX("C980"),9:REM Code fonction FMS : 9=lecture
350 ER%=USR(0):if ER%<>0 THEN PRINT "ERREUR";ER%:END
380 REM Le secteur est contenu entre $C9C0 et $CABF (256
octets).
```

Le début de ce programme SBASIC place en mémoire le programme assembleur. On demande à l'utilisateur le numéro de piste/secteur à lire (voir le guide de programmation avancée du Flex pour plus d'informations) qui doit être valide puisqu'aucun contrôle n'a été programmé .On stocke ces données dans la zone de passage des paramètres du FCB utilisé. En ligne 300, on fournit au SBASIC l'adresse de début du sous-programme assembleur qui est exécuter grâce à l'appel USR(0) en 350.

On remarque que l'on ne fournit pas de paramètre au sous-programme assembleur, mais que le code erreur disque est retourné en $26/$27.

Ces dernières lignes, affiche le contenu du secteur sous la forme d'une ligne de 256 caractères ASCII consécutifs :

```
400 AD=HEX("C9C0"):FOR I%=0 TO 255:OCTET%=PEEK(AD+I%)
450 IF OCTET%>=32 AND OCTET%<127 THEN PRINT CHR$(OCTET%);:GOTO
480
460 PRINT "_";
480 NEXT I%:PRINT
500 END
```

14.1. APPEL PAR USR DE PLUSIEURS SOUS-PROGRAMMES

Bien qu'il n'y ait qu'une seule commande USR en SBASIC, il est possible d'utiliser, toutefois, plusieurs sous-programmes externes écrits en langage machine. Deux méthodes sont possibles :
- la première est de considérer l'argument de la fonction USR comme un code de commande qui servira d'aiguillage de traitement dans le sous-programme assembleur : USR(CODE%) où CODE% sera le code commande ;
- la seconde, consiste à changer l'adresse d'exécution du sous-programme (en $24/$25) avant chaque appel d'une fonction USR.

Si l'on dispose de 2 programmes assembleurs, l'un à l'adresse $C500, l'autre en $C600, ils pourront être appelés alternativement comme dans cet exemple :

```
100 DPOKE HEX("24"), HEX("C500"):REM Adresse début 1er progr.
200 PARA1%=USR(0):REM Exécute 1er programme.
210 DPOKE HEX("24"), HEX("C600"):REM Adresse début 2e progr.
220 PARA2%=USR(0):REM Exécute 2e programme.
```

15. EXECUTION DU SBASIC

Pour exécuter un programme sous SBASIC, FLEX-9 doit être actif et les trois plus (+++) affichés à l'écran.
Après quelques secondes SBASIC répond "PRET" pour indiquer qu'il est prêt à accepter des commandes ou l'introduction d'un programme. Si SBASIC est déjà présent en mémoire, le point d'entrée à froid ($100) le réinitialise. Le point d'entrée à chaud ($103) permet de retrouver un programme existant.

La commande de chargement de SBASIC peut être utilisée avec la syntaxe plus générale:

SBASIC (,<nom de progr>)

où "nom de progr" est le nom d'un fichier contenant un programme SBASIC qui sera chargé et exécuté. L'unité de chargement est, par défaut, l'unité de Travail, l'extension par défaut étant .BAC, c'est-à-dire un Programme BAsic Compilé. Pour charger et exécuter un programme BASIC écrit en langage source il faut préciser l'extension .BAS.

On peut placer dans le fichier STARTUP.TXT, l'ordre d'exécution direct d'un programme SBASIC qui sera chargé à l'initialisation du système (voir la documentation FLEX-9), par exemple :

```
TTYSET PS=N:SBASIC 0.PROGR.BAC
```

16. QUELQUES ADRESSE UTILES

SBASIC occupe environ 22 K de mémoire sous FLEX-9.
Les adresses utiles sont les suivantes :

MEMEND :

La plus haute adresse de la mémoire RAM utilisée par le SBASIC est précisée dans le FLEX-9 à l'adresse $CC2B ET $CC2C. C'est cette adresse qu'il faut modifier, avant de charger SBASIC, pour réserver de l'espace pour des sous-programmes externes appelés par la fonction USR. La procédure est la suivante : sauvegarder le programme courant s'il y en a un, sortir de SBASIC par la commande EXIT pour modifier MEMEND sous moniteur, réexécuter SBASIC au point d'entrée à froid $100.

COLD :

C'est l'adresse $100, point d'entrée à froid du SBASIC pour les différentes initialisations.

WARM :

C'est l'adresse $103, point d'entrée à chaud normalement utilisé après sortie de SBASIC par les commandes EXIT ou FLEX-9 pour préserver le programme courant. Le pointeur de pile ne doit pas être modifié.

EXIT :

C'est l'adresse $106, utilisé par SBASIC pour la commande EXIT. Il doit y trouver l'adresse du point d'entrée à chaud du moniteur en ROM.

17. SOMMAIRE DES ERREURS

Chaque fois qu'une erreur se produit en cours d'exécution d'un programme, l'exécution est arrêtée et un message d'erreur est affiché (sauf quand la procédure ON ERROR GOTO est active). Le message d'erreur peut être affiché en toutes lettres, (dans ce cas, la possibilité d'appel au fichier des messages ne doit pas être inhibée). Le numéro de la ligne qui a provoqué l'erreur et le numéro de l'erreur est indiqué dans tous les cas.

Les codes d'erreur de 1 à 49 sont relatifs aux entrées-sorties. Elles peuvent être gérées par le programme utilisateur au moyen de la procédure ON ERROR GOTO. Les erreurs signalées par FLEX-9 portent les numéros 1 à 29, celles relatives à la syntaxe des instructions ou au type des variables manipulées dans les expressions sont répertoriées entre 50 et 99, celles pouvant survenir dans les calculs arithmétiques sont codées de 101 à 109. Les erreurs 127 et 128 sont des erreurs système particulières, exclues de tout traitement par ON ERROR GOTO.

LISTE DES ERREURS

******** ERREURS SBASIC ********

No Erreur Correspondance

1 = Code de Fonction FMS Illegal
2 = Le Bloc de controle de fichier (FCB) specifie est deja utilise
3 = Le Fichier specifie existe deja
4 = Fichier inexistant, non trouve
5 = Erreur dans le Catalogue ou dommage, Recharger le Systeme
6 = Plus d'espace disque disponible pour le Catalogue
7 = Tout l'espace disque a ete utilise
8 = Fin de Fichier rencontree en lecture (ou ecriture)
9 = Erreur de Lecture sur Disque
10 = Erreur d'Ecriture sur Disque
11 = Fichier ou Disque Protege contre l'Ecriture ou l'Effacement
12 = Fichier Protege, Acces non autorise
13 = Bloc de Controle Fichier (FCB) Illegal
14 = Adressage du Disque Illegal
15 = No. d'Unite Disque Illegal
16 = Disque absent ou Lecteur Disque non pret
17 = Le Fichier est protege, acces refuse
18 = Violation des Attributs d'acces a un Fichier ou Fichier endommage
19 = Pointeur d'acces direct errone
20 = FMS Inactif ou endommage, Recharger le Systeme
21 = Specifications Illegales du Fichier
22 = Erreur systeme en Fermeture d'un Fichier
23 = Debordement table d'allocation, disque trop segmente (Disque a recopier avec COPY.CMD)
24 = No. d'Enregistrement inexistant (jamais ecrit ou inferieur ou egal a 0)
25 = Erreur No. d'enregistrement, Fichier endommage ou Lecteur defectueux
26 = Erreur de syntaxe, reconnez la commande FLEX correctement
27 = Imprimante non disponible (commande non autorisee pendant l'impression simultanee : Spool)
28 = Configuration Materiel Insuffisante
29 = Fichier vide ou indefini
30 = Types de donnee errones (Emploi d'Alpha dans une zone numerique. Sur INPUT ou READ)
31 = Fin de liste de DATA dans l'instruction READ/DATA
32 = Mauvais Argument dans une Instruction ON... GOTO ou ON... GOSUB (=0)
33 =
34 = Arret sur frappe d'un (CTRL) /C : arret du Programme SBASIC demande
36 =
37 = Arret sur frappe de (TTYSET SS) puis (RC) : arret du programme
38 =
39 =
40 = Erreur de Canal Fichier
41 = Fichier deja Ouvert

LISTE DES ERREURS (suite)

42 = Le Fichier doit etre ouvert en NEW ou OLD
43 = Le Fichier est deja ferme ou n'a jamais ete ouvert
44 = Erreur sur l'Etat du Fichier
45 = Taille trop Grande dans un FIELD (Taille de l'enregistrement est superieure a 252 caracteres)
46 = Un Fichier Secuentiel ne peut etre etendu
46 = Taille du tampon de l'enregistrement trop longue dans SIZE
47 = No. d'Enregistrement Illegal (Inferieur ou egal a 0)
48 = Ne s'applique que pour un Fichier a Acces Direct dans GET, PUT
49 = Enregistrement verrouille
50 = Instruction Non Reconnue ou Inexistante
51 = Caractere ou Signe Illegal dans la Ligne
52 = Erreur de Syntaxe
53 = Terminaison de la Ligne Illegale
54 = Numero de Ligne Illegal (Nulle ou inferieur a 0)
55 = Parentheses non Refermee ou Balancees
56 = Reference Illegale (ou non declaree) a une Fonction
57 = Guillemet (") manquant (ou superflu) dans une constante Chaine Alpha
58 = THEN ou GOTO manquant dans une Instruction IF
59 = SUB sans CALL
60 = No. de Ligne non trouvee (ou inexistante apres ON ERROR GOTO)
61 = RETURN sans GOSUB ni CALL
62 = Erreur dans les Boucles FOR-NEXT
63 = CONTinuation de l'Execution Impossible
64 = Programme Source Absent (il s'agit d'un programme Compile)
65 = Programme ne pouvant etre charge (Type .BAC incompatible : Utiliser le BASIC requis)
66 = RESUME absent de la sous-routine d'Erreur ON ERROR GOTO
67 = Facteur d'Echelle (SCALE) non modifiable
68 = L'extension du Fichier est illegale
69 =
70 = Erreur de Type dans un PRINT USING
71 = Format Illegal dans un PRINT USING
72 = Emploi de l'Alpha Illegal pour une Expression Numerique ou Booleenne
73 = Expression, Argument, Indice ou operateur non Conforme/Illegal
74 = Argument en Dehors de la Fourchette de Valeur (0 a 255)
75 = Argument en Dehors de la Fourchette de Valeur (-32768 a +32767)
76 = Type de Variable Illegal
77 = Reference a un Indice depassant les dimensions d'un Tableau
78 = Reference a un Tableau Non Declare/Dimensionne avec DIM
79 = Mauvais Argument dans une Instruction SWAP (types de donnees non appropriees)
80 = Debordement de Memoire Centrale
81 = Debordement de Tableau
82 = Debordement Memoire programme : taille insuffisante (augmenter l'argument de OVERLAY)
83 = Chaine Alpha trop Longue
84 = Trop d'Arguments dans SUB et LOCAL

LISTE DES ERREURS (fin)

85 = Chaine Vide dans l'Instruction EXECUTE ou mauvais arguments dans DELETE
86 = Chaine Trop Longue dans l'Instruction EXECUTE
87 = Pointeur de ligne errone au chargement - Mauvais fichier
88 = Redimensionnement d'un tableau non autorise, tableau a effacer au prealable (avec CLEAR)
89 = Effacement d'une ligne precedent la ligne courante
90 = Fonction USER non Definie
91 = Appel USER non Defini
92 = Argument Virtuel dans les Instructions SWAP ou SET
93 = SUB sans CALL
94 = Mauvaise Longueur de Chaine Specifiee
95 = LOCAL sans SUB
96 = Liberation Memoire par CLEAR d'un Tableau Virtuel non Referme
97 = Positionnement du Curseur avec de mauvais arguments
98 = Mauvais arguments dans l'instruction CLEAR
99 = Type errone dans les arguments d'un CALL
100 = Expression Trop Complexe
101 = Debordement de l'Expression Numerique dans une operation flottante
102 = Argument ou exposant Trop Grand dans EXP
103 = Division par Zero
104 = Reel Trop Grand pour une Conversion en Entier
105 = Argument Negatif ou Nul pour LOG
106 = Debordement Numerique d'une Variable, No.de Ligne ou dans INPUT
107 = Racine Carree d'un nombre negatif
108 = Erreur de Conversion (Nombre trop Grand de plus de 16 chiffres)
109 = Debordement dans une Operation Entiere
110 = Numero de Couleur Superieur a 7
111 = No de PORT Impossible
112 = Numero de Manette de Jeu Video Illegal
113 = -
114 = Editeur de Ligne : Ligne non Trouvee ou Vide dans EDIT
115 = Editeur de Ligne : No. de Ligne Invalide ou Inexistante dans EDIT
116 = -
117 = -
118 = Editeur de Ligne : Ligne Trop Longue
119 = Editeur de Ligne : No. de Ligne Absent
120 = Editeur de Ligne : Instruction Absente
121 = Numero de PORT Illegal

127 = Erreur non Recuperable
128 = Erreur non Recuperable

255 = Erreur Systeme

INDEX DES INSTRUCTIONS, DES FONCTIONS ET DES COMMANDES

DIFFERENCES ENTRE LES VERSIONS SBASIC V2.33 FLEX ET SBASIC V2.37 MSDOS - CPM86

Ces différences sont dues aux systèmes d'exploitation :

1) FONCTIONS NON IMPLANTEES :

* POS
* USR
* DATE$

2) INSTRUCTIONS NON IMPLANTEES :

* EXEC

3) COMMANDES NON IMPLANTEES :

* +

Les restrictions 2) et 3) proviennent du fait qu'on ne peut exécuter une commande SYSTEME sous SBASIC dans la version CPM86.

Pour pallier aux inconvénients découlant de ces restrictions, des commandes BASIC supplémentaires ont été ajoutées. Dans la version MSDOS - CPM86, les utilitaires FLEX deviennent des commandes SBASIC.

4) COMMANDES AJOUTEES :

* TYPE "Nom fichier"

- Liste le fichier nommé sur l'écran - "Nom fichier" est un nom de fichier au format CPM, ambigu ou non. Si la référence est ambiguë, le premier fichier en correspondance est listé.

Exemples :

TYPE "ESSAI.BAS" liste le fichier ESSAI.BAS du disque courant ;
TYPE "B:*.TXT" liste le premier fichier d'extension .TXT du disque "B:".

* DIR "Nom fichier"

- liste la "directory" en correspondance avec "Nom fichier".

Exemples :

DIR "*.*" liste toute la directory du disque courant ;
DIR "B:T*.BAS" liste les noms des fichiers commençant par la lettre T, d'extension .BAS, et se trouvant sur le disque "B:".

* RENUMBER

Action identique à l'utilitaire SBRENUMB. Syntaxe identique.

* COMPILE "NOM fichier", ...etc...

Action identique à l'utilitaire COMPILE, à la différence près que le nom du fichier doit figurer entre guillemets (quotes).

* TRIVAR

Action identique à l'utilitaire TRIVAR.

* REFSUB

Action identique à l'utilitaire REFSUB.

L'utilitaire TASVAR n'est pas reconduit.

5) SUR LES NOMS DE FICHIERS :

Un nom de fichier peut être donné sous deux formes :
- soit la forme MSDOS-CPM86 : A:ESSAI.TXT
- soit la forme FLEX : 1.ESSAI.TXT ;

avec la correspondance :

1 pour A, 2 pour B, ...etc...

6) OUVERTURE DU CANAL 0 EN IMPRESSION :

La syntaxe OPEN "O.PRINT" AS 0 est reconnue, mais le nom de fichier "O.PRINT" n'est pas pris en compte. Il peut notamment ne pas exister, mais son nom doit être syntaxiquement correct.

7) MANIPULATION DES FICHIERS A ACCES DIRECT :

Deux différences essentielles apparaissent :

- Un fichier ouvert en OPEN NEW est à accès séquentiel en sortie (écriture séquentielle).
- Un fichier ouvert en OPEN OLD est à accès séquentiel en entrée (lecture séquentielle).
- Un fichier ouvert en OPEN est à accès direct (tableau virtuel ou enregistrements).

Ensuite,

- L'ouverture effective d'un fichier séquentiel a lieu lors de la première entrée/sortie exécutée sur le fichier. L'absence éventuelle du fichier sera détectée à ce momment là.
- L'ouverture effective d'un fichier à accès direct a lieu lors de la première spécification du fichier par l'une des déclarations suivantes :

FIELD# ou DIM#

L'instruction FIELD#,...etc... doit donc être exécutée avant toute opération GET# ou PUT# sur le fichier.

Enfin :

- Il est possible pour les fichiers à accès direct de spécifier la longueur de l'enregistrement, par exemple :

```
OPEN "EMPLOYE" AS 1 , LEN 1000
```

ouvre le fichier EMPLOYE.DAT du disque courant avec des enregistrements de 1000 octets.

La déclaration de longueur est optionnelle. La valeur par défaut est 252 pour rester compatible avec la version FLEX.

ATTENTION ! La création de l'enregistrement 50, par exemple, par l'instruction PUT#<canal>,record 50, ne crée pas les enregistrements précédents sous MSDOS et CPM86, contrairement au FLEX. Par conséquent la lecture d'un enregistrement de numéro inférieur non créé, provoquera une erreur.

Pour terminer signalons que toutes les opérations graphiques et les ports sont opérationnels grâce au moniteur G3MON88, qui est de ce point de vue, très complet.

NOTES

NOTES

Imp. Landais - 93160 Noisy-le-Grand - R.C. Pontoise B 572.170.660