

PROGRAMMATION
EN
APPLESOFT

sonotec

PROGRAMMATION EN APPLESOFT

TABLE DES MATIERES

INTRODUCTION

EXPRESSIONS

- 2.1 Constantes numériques
- 2.2 Limites de valeur des constantes numériques
 - 2.2.1 Représentations de zéro
 - 2.2.2 Limitations dues à la représentation interne
- 2.3 Constantes alphanumériques
- 2.5 Variables
 - 2.5.1 Noms de variables
 - 2.5.1.1 Variables réelles
 - 2.5.1.2 Variables entières
 - 2.5.1.3 Variables alphanumériques
 - 2.5.1.4 Effet des erreurs sur les noms
 - 2.5.2 Limites des valeurs des variables réelles
 - 2.5.3 Limites des valeurs des variables entières
 - 2.5.4 Constitution des variables alphanumériques
 - 2.5.5 Variables indicées
- 2.6 Fonctions
- 2.7 Opérateurs
 - 2.7.1 Opérateurs arithmétiques
 - 2.7.2 Opérateurs relationnels
 - 2.7.3 Opérateurs logiques ou booléens
 - 2.7.4 Opérateurs de concaténation de chaînes de caractères
- 2.8 Règles d'évaluation des expressions
 - 2.8.1 Rôle des parenthèses
 - 2.8.1 Hiérarchie des opérateurs
- 2.9 Convention de notation

INSTRUCTIONS

3. Commentaires

3.1 REM

4. Instruction de définition de tableaux et de fonctions

4.1 DIM

4.2 DEF FN

5. Instructions d'assignation de valeurs

5.1 LET

5.2 POKE

6. Instructions de contrôle du déroulement d'un programme

6.1 GOTO

6.2 ON ... GOTO

6.3 IF ... THUN

6.4 IF ... GOTO

6.5 FOR

6.6 NEXT

6.7 Sous programmes

6.7.1 GOSUB

6.7.2 ON ... GOSUB

6.7.3 RETURN

6.7.4 POP

6.7.5 CALL

6.8 ON ERR GOTO

6.9 RESUME

6.10 END

6.11 STOP

6.12 WAIT

7. Les instructions d'entrée/sortie

7.1 INPUT

7.2 GET

7.3 PRINT

TAB

SPC

7.4 DATA

7.5 READ

7.6 RESTORE

7.7 STORE

7.8 RECALL

7.9 IN #

7.10 IN # (DOS)

7.11 PR #

7.12 PR # (DOS)

8. Commandes

8.1 LIST

8.2 DEL

8.3 RUN

8.4 CTRL G

8.5 CONT

8.6 TRACE

8.7 NOTRACE

8.8 JOAD

8.9 SAVE

8.10 NEW

8.11 CLEAR

8.12 &

9. Instructions de mise en page des affichages

9.1 HTAB

9.2 VTAB

9.3 HOME

9.4 FLASH

9.5 INVERSE

9.6 NORMAL

9.7 SPEED

10. Instructions de graphisme et de jeux

10.1 TEXT

10.2 GR

10.3 COLOR

10.4 PLOT

10.5 HLIN

10.6 VLIN

10.7 SCRN

10.8 HGR

10.9 HGR2

10.10 HPLOT

10.11 PDL

11. Fonctions mathématiques et traitement des chaînes alphanumériques.

INTRODUCTION

1.1. MODES D'EXECUTION D'UN PROGRAMME APPLESOFT.

Le BASIC APPLESOFT reconnaît deux modes d'exécution des programmes :

- le mode immédiat où les instructions sont exécutées dès qu'elles ont été prises en compte par le système
- le mode différé où les instructions stockées en mémoire sont exécutées lors d'une demande explicite

1.1.1 MODE IMMEDIAT.

Un programme en mode immédiat est une série d'instructions BASIC APPLESOFT qui ne commence pas par une série de chiffres qui serait interprétée comme un numéro de ligne et en conséquence, la série d'instruction serait stockée pour utilisation ultérieure. La série d'instruction prise en compte par le système est l'ensemble des caractères effectivement fournis au système (en tenant éventuellement compte des corrections faites lors de la frappe) entre le moment où BASIC a envoyé le caractère d'identification] et le moment où l'opérateur a frappé la touche RETURN. L'assemblage des instructions à l'intérieur de cette chaîne de caractères suit les règles de syntaxe normales.

1.1.2 MODE DIFFERE

Un programme en mode différé est constitué d'une série de lignes numérotées, chaque ligne étant constituée d'une série d'instructions se conformant aux règles de syntaxe du BASIC APPLESOFT. Le contenu d'une ligne est conservé par le système jusqu'au moment où on la supprime explicitement. Un programme écrit pour une exécution en différé peut être conservé sur disque ou sur cassette, ce qui n'est pas possible en mode immédiat.

1.2 FORMAT DES INSTRUCTIONS

Les instructions servent à écrire le programme BASIC qui sera exécuté ultérieurement. Chaque instruction remplit une fonction précise et est constituée de mots-clés et de paramètres. Les instructions constituant un programme sont séparées sur une même ligne par le symbole : et la séparation entre les lignes est opérée par la frappe de la touche RETURN.

Les instructions BASIC sont en format libre, c'est à dire que les blancs n'ont pas de signification et sont ignorés sauf dans certains cas particuliers qui seront examinés chaque fois qu'ils se présenteront.

Ainsi les différentes instructions suivantes sont équivalentes :

] 30PRINTS

] 3 0 P R I N T S

] 30 PRINT S

1.3 CONVENTIONS DE SYNTAXE

Les conventions de syntaxe suivantes sont utilisées dans la suite de ce manuel.

MAJUSCULES

Ces caractères doivent être fournis au système de la façon spécifiée.

minuscules italiques

Elles indiquent le type d'information que doit fournir l'utilisateur. C'est le cas en général des paramètres.

*(paramètre 1)
(paramètre 2)
(paramètre 3)*

Un et un seul paramètre est à fournir parmi la liste

(paramètre)

paramètre optionnel

. . .

Les points de suspension indiquent que les précédents paramètres sont répétés

EXPRESSIONS

Une expression combine des constantes, des variables ou des fonctions avec des opérateurs dans une séquence ordonnée. Une expression doit fournir un résultat lors de son évaluation. Les constantes, les variables et les fonctions représentent des valeurs ; les opérateurs indiquent à la machine quelles opérations effectuer sur ces valeurs.

Voici deux exemples d'expressions :

$(P + 5)/27$

P est une variable à laquelle on doit avoir préalablement donné une valeur. 5 et 27 sont des constantes. / est l'opérateur de division. Les parenthèses groupent les expressions qui doivent être évaluées en premier.

Si $P = 49$, l'expression est évaluée à 2.

$(N - (R + 5)) - T$

N, R et T sont des variables auxquelles on doit avoir préalablement assigné une valeur. + et - sont les opérateurs d'addition et de substraction. Les parenthèses intérieures entourent la partie à évaluer en premier.

Si $N = 20$, $R = 10$, $T = 5$, alors l'expression est évaluée à 0.

2.1 CONSTANTES NUMERIQUES.

Une constante numérique est un nombre décimal négatif, positif, ou nul. Il peut être exprimé sous les trois formes suivantes :

- sous forme de nombre entier, c-a-d une série de chiffres sans point décimal ;
- sous forme d'un nombre en virgule fixe, c-a-d une série de chiffres précédés, séparés ou suivis d'un point décimal.
- sous forme d'un nombre en virgule flottante, c-a-d d'un nombre en virgule fixe ou entier suivi de la lettre E et d'un nombre entier éventuellement signé.

Les constantes numériques sont toujours considérées par le BASIC APPLESOFT comme des nombre réels, dont la représentation interne est celle d'un nombre réel en virgule flottante.

Dans la notation en virgule flottante, le nombre précédant la lettre E est appelée mantisse, le nombre suivant la lettre E est appelé exposant et l'exposant est la puissance de 10 par laquelle on doit multiplier la mantisse pour obtenir la valeur du nombre considéré.

Par exemple,

1E23	$= 1 \times 10^{23} = 100000000000000000000000$
1.0E23	comme ci-dessus
.001E26	comme ci-dessus
1.02E + 4	$= 1.02 \times 10^4 = 10200$
1.02E-4	$= 1.02 \times 10^{-4} = .000102$

2.2 LIMITES DES VALEURS DES CONSTANTES NUMERIQUES.

Les constantes numériques sont limitées en magnitude par le format de la représentation interne des nombre réels.

Une constante dont la valeur absolue est inférieure à 2.9388.E -39 est égale à 0.

Une constante dont la valeur absolue est supérieure à 1.E 38 cause l'apparition du message d'erreur ?OVERFLOW ERROR c'est à dire dépassement de capacité.

2.2.1 REPRESENTATIONS AUTORISEES DE ZERO.

Le BASIC APPLESOFT admet de nombreuses représentations du nombre zéro. Les expressions suivantes sont évaluées à 0 :

	+	-	.E	+.E	-.E
.E +	.E-	+.E-	+.E +	-.E +	-.E-

Ainsi l'élément d'un tableau TAB (.) est le même que celui représenté par TAB (0).

2.2.2 LIMITATIONS DUES A LA REPRESENTATION INTERNE.

Un nombre réel occupe une place limitée en mémoire (5 octets). Le nombre de chiffres significatifs qui le décrivent est donc limité. L'interpréteur BASIC APPLESOFT opère donc un arrondissement des nombres dont le nombre de chiffres significatifs est supérieur à la précision de la représentation interne. Celle-ci est de $1.8 \cdot 10^{-12}$ au mieux, ce qui permet d'avoir facilement une précision de 9 chiffres significatifs (et même 10 entre 0.01 et 0.1) après les traitements internes des nombres réels. Néanmoins l'opération d'arrondi fait parfois apparaître des résultats curieux. Si la constante a plus de 9 chiffres (même égaux à 0), il risque de se produire une troncature et non une opération d'arrondi. Ainsi on peut être surpris des résultats suivants :

```

]PRINT 999999999.9
99999999.9
]PRINT 999999999.90
100000000
]PRINT 11.11111145000
11.1111115
]PRINT 11.1111114519
11.1111114

```

On prendra garde, lors de l'écriture des programmes, à ne pas utiliser des constantes contenant plus de 38 chiffres sous peine de voir apparaître le message ?OVERFLOW ERROR, même dans le cas où tous les chiffres derrière le point décimal sont des 0. Néanmoins dans le cas où le premier chiffre est un 1 et le second inférieur ou égal à 6, la machine accepte des nombres de 39 chiffres.

Il est néanmoins à remarquer que de telles constantes sont sans signification pour la machine, étant donné la précision à laquelle on peut prétendre pour des nombres réels. La précision des nombres réels en BASIC APPLESOFT est de 9 à 10 chiffres significatifs. Il est donc superflu de définir des constantes avec plus de 10 chiffres significatifs.

2.3 CONSTANTES ALPHANUMERIQUES.

Une constante alphanumérique consiste en une chaîne de caractères délimitée par des cotes ``.

Une constante alphanumérique peut contenir entre les cotes n'importe quel caractère ASCII, qu'il puisse être imprimé ou non, à l'exception de la cote ``.

La chaîne de caractère vide ou nulle s'exprime par la constante `` ``.

A l'intérieur des cotes, les blancs sont significatifs.

Divers exemples ci-dessous illustrent la notion de constante alphanumérique :

```

``abc``
`` ABC ``

```


"CIEL ! MES BIJOUX ? ! "

""

" "

chaîne alphanumérique nulle

chaîne alphanumérique constituée de deux blancs.

Dans le cas où la constante alphanumérique est la dernière donnée introduite sur une ligne de programme, la cote finale peut être omise, le retour du curseur provoqué par l'action sur la touche RETURN s'y substituant.

2.5 VARIABLES.

Une variable est un nom auquel est assigné une valeur. Cette valeur peut être modifiée pendant le déroulement du programme. Une référence dans l'énoncé du programme à une variable agit lors de l'exécution comme une référence à sa valeur.

Le BASIC APPLESOFT reconnaît trois types de variables :

- les variables entières
- les variables réelles
- les variables alphanumériques.

Chacun de ces trois types de variables est lui-même subdivisé en variables simples et en variables indicées.

2.5.1 NOMS DE VARIABLES.

2.5.1.1 VARIABLES REELLES.

Le nom d'une variable réelle doit impérativement suivre les règles suivantes :

- commencer par une lettre
- être composé exclusivement de chiffres et de lettres
- ne pas contenir de mots réservés du BASIC APPLESOFT dont la liste est donnée en annexe.
- avoir une longueur au plus égale à 238.

Le BASIC APPLESOFT ne considère comme significatifs que les **deux premiers** caractères du nom de la variable. Ainsi, dans le cours d'un programme, toute référence à une variable nommée VARIA sera identique à une référence à une variable nommée VA.

Voici des exemples de nom de variable accepté par le BASIC APPLESOFT :

X

X1123EFT

Les noms de variables réelles suivants sont incorrects :

1A

AX2%

Z("

XLIST

FEND

Le premier caractère est un chiffre.

Le nom contient des caractères différents de lettres ou de chiffres.

Le nom contient en son corps un mot réservé par le BASIC APPLESOFT.

2.5.1.2 VARIABLES ENTIERES.

La formation d'un nom de variable entière suit les mêmes règles de formation qu'une variable réelle sauf pour le dernier caractère qui doit être **obligatoirement** un symbole %.

Les noms de variables entières suivants sont corrects :

A%

X67ERT%

Les noms de variables entières suivants sont incorrects :

1A%

AXI

XLIST%

Le premier caractère est un chiffre.

Le nom ne se termine pas par %.

Le nom contient un mot réservé du BASIC APPLESOFT.

2.5.1.3 VARIABLES ALPHANUMERIQUES.

La formation d'un nom de variable alphanumérique suit les mêmes règles de formation que les variables réelles sauf pour le dernier caractère qui doit être **obligatoirement** un symbole dollar \$.

Les noms de variables alphanumériques suivants sont corrects :

A\$

AX34567RT\$

Les noms de variables alphanumériques suivants sont incorrects :

1A\$

AX345

FEND\$

XLIST\$

Le premier caractère n'est pas une lettre.

Le dernier caractère n'est pas \$.

Le nom contient en son corps un mot réservé du BASIC APPLESOFT.

2.5.1.4. EFFET DES ERREURS SUR LES NOMS.

Bien qu'il ne soit pas possible de faire une liste exhaustive de tous les cas qui peuvent se présenter (puisque la détection des erreurs se fait, en BASIC APPLESOFT, uniquement lors de l'exécution des instructions), on peut signaler les cas les plus fréquents.

Si l'on met un chiffre en tête d'un nom de variable, des phénomènes bizarres apparaîtront : 1000 1A = 10 sera interprété par BASIC APPLESOFT comme 1001 A = 10, ce qui a pour effet de créer une ligne 10001 dans le programme alors qu'on voulait composer la ligne 1000.

Dans le cas où un chiffre ou un caractère différent d'une lettre commence un nom de variable dans le cours d'une instruction, on obtiendra en général le message ?SYNTAX ERROR. Par exemple, l'instruction suivante provoquera ce message : 1000 IF 1A = 0 GOTO 1200

Dans le cas où un mot réservé du BASIC APPLESOFT fait partie du nom de la variable, deux cas peuvent se produire. Le plus fréquent sera celui où apparaîtra le message 0SYNTAX ERROR. Par exemple, l'instruction suivante est dans ce cas : 1000 XLIST = 1

Dans d'autres circonstances, il y aura exécution du mot réservé et les résultats dépendent alors du mot et de son contexte.

Dans le cas où une variable alphanumérique n'est pas terminée par le symbole \$, on aura l'apparition du message ?TYPE MISMATCH ERROR car l'interpréteur BASIC APPLESOFT s'attend à trouver une valeur numérique pour valeur de la variable. C'est le cas dans l'instruction suivante :

```
1000 A = " 123 "
```

2.5.2 LIMITES DES VALEURS DES VARIABLES REELLES.

Les variables réelles comme les constantes numériques ont une représentation interne qui permet de manipuler des grandeurs dont la valeur est comprise entre $-1E\ 38$ et $1\ E\ 38$. Si les calculs sont des additions ou des soustractions, on peut atteindre une valeur absolue de $1.7\ E\ 38$. Au delà, il apparait le message d'erreur ?OVERFLOW ERROR.

Les variables dont la valeur absolue est inférieure à $2.9388\ E-39$ sont converties par la machine en zéro.

2.5.3 LIMITES DES VALEURS VARIABLES ENTIERES.

Les variables entières sont comprises en valeur entre -32767 et 32767 . Lors de la conversion de nombres réels en nombre entiers, le BASIC APPLESOFT opère en général une troncature, convertissant le nombre réel en le nombre entier qui lui est immédiatement inférieur.

```
A% = 1.5
PRINT A%
1
A% = -1.5
PRINT A%
-2
```

Néanmoins, dans le cas d'égali-sation avec une constante réelle, les problèmes d'arrondis bizarres se posent de nouveau, mais ce sont en fait des problèmes liés à la représentation des nombres réels.

Ainsi la constante réelle 123.99999996 étant évaluée à 124, il ne faut pas s'étonner des résultats des instructions suivantes :

```
]A% = 123.99999996
]PRINT A%
124
```

2.5.4 CONSTITUTION DES VARIABLES ALPHANUMERIQUES.

Une variable alphanumérique est constituée d'une chaîne de caractères quelconques d'une longueur inférieure ou égale à 255. Si, au cours de l'exécution d'un programme, la longueur d'une variable alphanumérique dépasse 255, le message ?STRING TOO LONG ERROR apparaît et la chaîne est trompée.

2.5.5 VARIABLES INDICEES

Les variables indicées sont des variables qui sont rangées dans des tableaux et dont l'indice permet de repérer leur position à l'intérieur d'un tableau.

Le nom du tableau est alors le nom de la variable, l'indice étant le numéro de la variable. Ainsi, si A est un tableau, A(0) représente le premier élément de ce tableau, A(5) son sixième élément.

Le BASIC APPLESOFT reconnaît qu'il a affaire à un tableau quand le nom de la variable est suivi d'un nombre ou d'une expression permettant d'évaluer ce nombre placé entre parenthèses.

Ainsi les noms de tableau suivent les mêmes règles de formation que les variables simples correspondantes :

A	est un nom de tableau réel
A\$	est un nom de tableau alphanumérique
A%	est un nom de tableau entier.

A l'intérieur d'un tableau, les variables sont stockées côte à côte comme le seraient des variables simples.

A des fins de simplicité d'expression, le BASIC APPLESOFT permet d'attribuer à une variable jusqu'à 88 indices qui servent à localiser la variable dans le tableau, ce qui équivaut à dire qu'un tableau peut lui-même être composé d'un ensemble de tableau jusqu'à concurrence d'un nombre de 89 ou **jusqu'à épuisement de la mémoire disponible**.

Le nombre d'indices qui définissent la position d'une variable à l'intérieur d'un tableau est appelé nombre de **dimensions** du tableau, la dimension étant le nombre de valeurs que peut prendre l'indice correspondant moins un.

Le BASIC APPLESOFT autorise aux indices de prendre des valeurs comprises entre 0 et la dimension correspondant au rang de l'indice. Cela signifie que si l'on a, par exemple 3 indices I, J et K, I ne pourra varier qu'entre 0 et la dimension qu'on a fixée pour le premier indice, K entre 0 et la dimension qu'on a fixée pour le troisième indice.

Si l'on ne fixe pas explicitement le nombre de dimensions et les valeurs de ces dimensions par l'ordre BASIC DIM (voir plus loin), l'interpréteur BASIC APPLESOFT fixe comme nombre de dimensions pour un tableau le nombre d'indices qu'il rencontre à la première référence faite à ce tableau, et fixe chacune des dimensions à la valeur de 10.

Le BASIC APPLESOFT autorise aux indices de prendre des valeurs non entières positives ou nulles inférieures ou égales à la dimension correspondante. Si les valeurs sont non entières, elles sont arrondies suivant les règles et avec les exceptions citées au paragraphe 2.5.3.

Si la valeur prise par un indice est négative, il apparait le message ?ILLEGAL QUANTITY ERROR.

Si la valeur d'un indice dépasse la dimension déclarée ou attribuée par l'interpréteur, il apparait le message ?BAD SUBSCRIPT ERROR.

Si les dimensions d'un tableau sont $N1, N2, N3, \dots, Np$, alors l'élément du tableau repéré par les indices $I1, I2, I3, \dots, Ip$ est l'élément du tableau situé à la $I1 + I2 \times N1 + I3 \times N1 \times N2 + \dots + Ip \times N1 \times \dots \times N_{p-1}$ ième place, (p étant le nombre de dimensions du tableau).

A étant un tableau, l'élément d'indices $i1, i2, \dots, ip$ est noté en BASIC APPLESOFT A (i1,i2, ..., ip).

La place occupée en mémoire par un tableau est proportionnelle au produit des dimensions. Il faut donc prendre garde de ne pas inutilement réserver de la place pour des variables indicées. Ainsi une variable réelle occupant 5 octets, il est impossible de déclarer un tableau de 10000 nombres réels, la mémoire de l'appareil étant limitée à un peu plus de 48 000 octets.

2.6 FONCTIONS.

Le BASIC APPLESOFT reconnaît trois types de fonctions :

- les fonctions réelles d'une variable réelle qui sont partie intégrante du langage BASIC APPLESOFT (voir la liste en 11)
- les fonctions réelles d'une variable réelle qui sont définies par l'utilisateur par l'ordre DEF (voir 4.2)
- les fonctions de traitement de chaînes de caractères (voir 11)

Une fonction étant une opération qui ne fournit qu'une seule valeur, elle peut être utilisée partout où l'on s'attend à voir une variable ou une constante. Le paramètre sur lequel opère la fonction peut être une expression quelconque, donc une variable, une constante, une variable indicée ou une fonction elle-même.

L'appel d'une fonction dans un programme se fait en faisant suivre le nom de la fonction par l'expression qu'elle doit évaluer située entre parenthèses.

Par exemple, SQR est le nom de la fonction BASIC qui calcule la valeur de la racine carrée d'un nombre positif. Donc, pour mettre dans la variable Y la valeur de la racine carrée de la variable X, il suffit d'écrire :

$$Y = \text{SQR} (X)$$

2.7 OPERATEURS.

Un opérateur exécute une opération logique ou mathématique sur une ou deux valeurs dont le résultat est une valeur unique. En général, un opérateur est situé entre deux valeurs, mais certains opérateurs, dits unaires, précèdent la valeur qu'ils affectent. Ainsi par exemple, le signe - dans l'expression A-B est un opérateur binaire dont le résultat est la différence entre A et B. Par contre dans l'expression - A, le signe - indique qu'il s'agit de considérer la valeur opposée de A.

La combinaison de un ou deux opérandes avec un opérateur forme une expression. Les opérandes qui peuvent apparaître dans une expression peuvent être des constantes, des variables, des fonctions ou des expressions elles-mêmes.

Les opérateurs sont divisés en types distincts suivant le genre d'opération qu'ils exécutent. Ce sont les opérateurs arithmétiques, relationnels, logiques ou Booléens, et l'opérateur de concaténation des chaînes de caractères.

2.7.1 OPERATEURS ARITHMETIQUES.

Les opérateurs arithmétiques sont :

<i>symbole</i>	<i>action</i>	<i>notation habituelle</i>
+	additionne ou sans effet	$A + B$ ou $+ A$
-	soustrait ou prend l'opposé	$A - B$ ou $- A$
*	multiplie	$A \times B$
/	divise	$A \div B$
.	élève à la puissance	A^B

Les opérateurs arithmétiques agissent sur des opérandes réels ou entiers. Le résultat est toujours un nombre réel.

2.7.2 OPERATEURS RELATIONNELS.

Les opérateurs relationnels sont :

<i>symbole</i>	<i>signification</i>	<i>notation habituelle</i>
=	égal à	$A = B$
>	supérieur à	$A > B$
<	inférieur à	$A < B$
= <	inférieur ou égal à	$A \leq B$
= >	supérieur ou égal à	$A \geq B$
> <	différent de	$A \neq B$

Quand on évalue une expression contenant des opérateurs relationnels, on attribue la valeur 1 à une relation qui est vraie et 0 à une relation qui est fausse. Par exemple, $A = B$ vaut 1 si A égale B, et 0 dans le cas contraire.

2.7.3 OPERATEURS LOGIQUES OU BOOLEENS.

Les opérateurs logiques ou booléens sont :

<i>symbole</i>	<i>signification</i>	<i>notation habituelle</i>
AND	produit logique de A et B	$A \wedge B$
OR	somme logique de A et B	$A \vee B$
NOT	négation ou complément de A	$\overline{A}$

Le produit logique de variables booléennes, c'est à dire de variables qui ne peuvent prendre que les valeurs 0 ou 1, est égal à leur produit au sens habituel du mot produit. En d'autres termes, le produit logique de A et B n'est vrai que si A et B sont vrais tous deux.

La somme logique de A et B est vraie, c'est à dire vaut 1, si l'un ou l'autre de A ou B est vrai ou si les deux sont vrais.

Le complément de A est vrai si A est faux, faux si A est vrai.

L'opérateur NOT étant unaire, on peut le composer avec AND et OR. On obtient alors des relations qui peuvent simplifier l'expression des conditions dans les programmes.

$$\begin{aligned} \text{NOT} (A \text{ AND } B) &= \text{NOT } A \text{ OR NOT } B \\ \text{NOT} (A \text{ OR } B) &= \text{NOT } A \text{ AND NOT } B \end{aligned}$$

Si A est effectivement une variable réelle, et non une expression, A est considérée comme vraie si A est différente de 0 et fausse si elle est égale à 0. Cette remarque peut permettre de diminuer le temps d'exécution de certains tests.

2.7.4 OPERATEUR DE CONCATENATION DE CHAINES DE CARACTERES.

L'opérateur de concaténation de chaînes de caractères en BASIC APPLESOFT est le symbole +

Ainsi "CECI EST U " + " N TEST " est la chaîne de caractères :
CECI EST UN TEST

2.8 REGLES D'EVALUATION DES EXPRESSIONS.

Une expression est évaluée en donnant à chaque variable la valeur qui lui est associée, en calculant les résultats des fonctions qui font partie de l'expression et en effectuant les opérations indiquées par les opérateurs. L'ordre dans lequel les opérations sont effectuées est déterminé par la hiérarchie des opérateurs et l'existence de parenthèses.

2.8.1 ROLE DES PARENTHESSES.

Lorsque les parenthèses ne sont pas présentes pour des raisons de syntaxe du BASIC (comme par exemple pour indiquer la présence d'indices ou pour délimiter les paramètres de fonctions), les opérations situées à l'intérieur des parenthèses sont exécutées avant celles situées à l'extérieur des parenthèses. Dans le cas où des parenthèses sont imbriquées, les opérations situées entre les parenthèses les plus intérieures sont exécutées les premières, puis celles situées entre les parenthèses d'un niveau d'imbrication inférieur.

Par exemple, dans l'expression : $1 + (2 + (3 + 4))$,
la première opération effectuée est l'addition $3 + 4$, puis l'addition $2 + 7$, et enfin l'addition $1 + 9$.

IL EST INDISPENSABLE QU'IL Y AIT TOUJOURS UN NOMBRE EGAL DE PARENTHESES OUVRANTES ET FERMANTES. DANS LE CAS CONTRAIRE, IL Y A ERREUR SIGNALEE PAR LE MESSAGE ?SYNTAX ERROR.

2.8.2 HIERARCHIE DES OPERATEURS.

Dans l'ordre de priorité décroissante, la liste des opérateurs est la suivante :

^	exponentiation
NOT + -	opérateurs unaires
* /	multiplication division
+ -	addition soustraction
Opérateurs relationnels	
AND	
opérateurs logiques	
OR	

Si deux opérateurs ont la même priorité, la première opération exécutée est la première rencontrée : les opérateurs sont traités de gauche à droite dans le sens de progression de l'instruction.

En raison du rôle joué par les parenthèses explicité au 2.8.1, cet ordre de priorité n'est valable que pour l'évaluation des expressions ou portions d'expressions sans parenthèses.

Par exemple : $5 + 6 * 7$ est évalué à $5 + (6 \times 7)$, soit 47
 $7/14 * 2/5$ est évalué à $((7/14) \times 2)/5$, soit 0.2

Soit $A = 1$, $B = 2$, $C = 3$.

$A + B * C$ est évalué à $A + (B * C) = 7$

$A * B + C$ est évalué comme $(A * B) + C = 5$

$A + B - C$ est évalué comme $(A + B) - C = 0$

Quand un opérateur unaire suit immédiatement un autre opérateur de priorité supérieure, l'opérateur unaire prend le rang de cet opérateur. Par exemple,

Soit $B = B^{\wedge} - B^{\wedge} C$ est évalué comme $(B^{\wedge} B)^{\wedge} C$, soit 1/64 ou .015625

Dans une expression logique, les opérateurs différents des opérateurs logiques sont d'abord considérés et si les opérandes des opérateurs logiques valent 0, ils sont considérés comme faux, et dans le cas contraire comme vrais.

Par exemple, NOT 1/2 est évalué à 0 donc faux car 1/2 est non nul donc vrai.

2.9 CONVENTION DE NOTATION.

Dans la suite de ce ce manuel, une expression sera notée

<i>exp</i>	si cette expression est quelconque
<i>exp num</i>	si cette expression prend des valeurs numériques
<i>exp alpha</i>	si cette expression a pour valeur une chaîne de caractères
<i>exp log</i>	si cette expression est vrai ou fausse.

INSTRUCTIONS

Les instructions que reconnaît le BASIC APPLESOFT permettent :

- de commenter un programme
- de définir les tableaux ou les fonctions de l'utilisateur
- d'assigner à une variable ou un élément de la mémoire une valeur
- de contrôler la séquence des instructions de façon conditionnelle ou systématique
- de gérer les entrées / sorties
- d'effectuer toutes les opérations possibles sur le contenu du programme
- de mettre en page sur l'écran, ou le dispositif d'impression, les résultats des programmes
- de tracer des courbes en basse résolution (40 × 48 points)
- de tracer des courbes en haute résolution (280 × 192 points)
- de gérer l'espace mémoire.

La description des instructions et leur syntaxe utilisent les conventions de notation exposées au 1.3 et 2.9.

Les fonctions que reconnaît BASIC APPLESOFT, en plus des fonctions de l'utilisateur, sont :

- les fonctions mathématiques usuelles
- les fonctions ayant trait au traitement des chaînes de caractères
- les fonctions spécifiques propres à la machine APPLE II.

La description de ces fonctions et leur syntaxe utilisent les conventions de notation exposées au 1.3 et 2.9.

3 COMMENTAIRES.

Les commentaires sont introduits dans un programme à l'aide de l'instruction REM.

3.1 REM

Objet : permettre l'inseton d'un texte quelconque dans le corps d'un programme.

Syntaxe : REM *chaîne de caractères*

Commentaires : La chaîne de caractères suivant l'ordre REM est totalement quelconque : elle peut contenir des cotes "", des séparateurs d'instruction : et tout caractère du jeu de caractères ASCII à l'exception du retour chariot, qui ne peut faire partie de la chaîne de caractères puisqu'il la termine nécessairement.

L'instruction REM termine donc toujours la ligne d'instruction sur laquelle elle est placée.

Modes d'exécution : immédiat et différé.

Exemples :

```
10 REM CECI EST UN COMMENTAIRE
20 REM COMMENTAIRE : PRINT " TEST "
```

L'exécution de ces lignes de programme ne se traduiront par aucun effet. L'instruction PRINT est ignorée et fait partie du commentaire.

4 INSTRUCTIONS DE DEFINITION DE TABLEAUX ET FONCTIONS.

Le BASIC APPLESOFT identifie au cours d'un programme toutes les variables qu'il rencontre et, suivant leur nom, réserve un emplacement mémoire pour ces variables. Pour les tableaux, par contre, l'interpréteur ignore la taille du tableau envisagé par l'utilisateur et c'est à ce dernier de préciser cette taille au moyen de l'instruction DIM.

Le BASIC APPLESOFT reconnaît comme fonctions définies par l'utilisateur les fonctions qui sont définies avant leur première utilisation par l'instruction DEF FN.

4.1 DIM

Objet : définir les dimensions d'un tableau et leur nombre.

Syntaxe : DIM *nom de variable* (*constante numérique* *expression numérique*, ...), ...

Commentaires : ● *nom de variable* doit respecter les règles de formation exprimées au 2.5.1 ;

● *constante numérique* *expression numérique* doit prendre une valeur non négative et théoriquement inférieure à 32767. En pratique, cette limite ne peut jamais être atteinte en raison de la taille limitée de la mémoire. Quand le tableau exige un espace mémoire supérieur à la taille de la mémoire disponible, il y a affichage du message d'erreur ?OUT OF MEMORY ERROR. Dans le cas où la valeur assignée à une dimension est négative, il y a affichage du message ?ILLEGAL QUANTITY ERROR.

● le nombre de dimensions d'un tableau est limité à 88, c'est à dire qu'on doit trouver moins de 89 virgules à l'intérieur des parenthèses les plus extérieures. Si l'on dépasse cette limite, il y apparition du message ?OUT OF MEMORY ERROR.

● l'instruction DIM doit précéder la première référence à *nom de variable* suivi de parenthèses. Dans le cas contraire, il y a apparition du message ?REDIM'D ARRAY ERROR. En effet, si la première référence à la variable indiquée de nom *nom de variable* n'est pas l'ordre DIM, l'interpréteur BASIC APPLESOFT affecte au tableau de nom *nom de variable* la valeur 10 pour chacune des dimensions. Par exemple, dans le programme suivant :

```
10 A(1,1,1) = 1
```

l'interpréteur va réserver par défaut 11 * 11 * 11 places pour le tableau A. (au sujet des dimensions, se reporter au 2.5.5). Le nombre de dimensions est calculé en comptant le nombre de virgules à l'intérieur des parenthèses moins un, c'est à dire le nombre d'indices.

● la valeur prise par le n^{ième} indice de *nom de variable* est la valeur maximum autorisée pour cet indice dans l'exécution du programme si cette valeur est entière. Si cette valeur n'est pas entière, elle est convertie en l'entier juste inférieur comme dans le cas des conversions nombre réel - nombre entier (voir cependant les commentaires en 2.5.3). Dans le cas où cette limite serait dépassée dans le cours d'un programme, il y aurait affichage du message ?BAD SUBSCRIPT ERROR.

● les chaînes de caractères ayant un format variable, elles occupent la mémoire disponible au fur et à mesure des besoins, c'est à dire que le programme : 10 A\$ = "ABC"

```
20 A$ = "D"
```

on trouvera en fait en mémoire ABSC et ce sont des pointeurs qui indiqueront si c'est à la chaîne ABC ou à la chaîne D que le programme se réfère. Pour repérer une chaîne, il suffit de connaître sa longueur et la position du premier caractère donc trois octets suffisent. Un tableau de variables alphanumériques est en fait un tableau de pointeurs et l'espace mémoire utilisé égal au produit des dimensions additionnées de 1 multiplié par 3, plus la longueur de toutes les chaînes de caractères effectivement utilisées.

● un nombre réel occupant 5 octets (un pour l'exposant, quatre pour la mantisse), un tableau de nombre réels occupe cinq fois le produit des dimensions additionnées de 1.

● un nombre entier occupant 2 octets, un tableau de nombres entiers occupe deux fois le produit des dimensions additionnées de 1.

● un schéma de l'occupation mémoire et la position des pointeurs se trouve en Appendice.

● on rappelle ici enfin que les variables simple A et indicée A (1, ..., ...) sont des variables distinctes.

Exemples :

10 DIM A(2,3,4) *réserve un tableau de 2 + 1 × 3 + 1 × 4 + 1 nombres réels soit 60 × 5 octets.*

10 DIM AX% (15) *réserve un tableau de 15 + 1 nombres entiers soit 32 octets.*

10 DIM A\$ (4,4) *réserve un tableau de 4 + 1 × 4 + 1 pointeurs de chaînes de caractères, soit 75 octets.*

10 A(2) = 127 *est un programme qui réserve pour la variable indicée un tableau de 10 + 1 valeurs réelles.*

4.2 DEF FN

Objet : permettre à l'utilisateur de définir une fonction d'une variable réelle, cette fonction prenant des valeurs réelles.

Syntaxe : DEF FN *nom de variable réelle 1* (*nom de variable réelle 2*) = *exp num*

Commentaires : ● la fonction définie par cet ordre a pour nom FN *nom de variable réelle 1*. La variable réelle située entre parenthèses de nom *nom de variable réelle 2* est appelée argument de la fonction. L'expression numérique notée *exp num* est l'expression qui permet d'évaluer la fonction de nom FN *nom de variable réelle 1*. Si cette expression fait intervenir la variable *nom de variable réelle 2*, le résultat des calculs exprimés par *exp num* vont effectivement dépendre des valeurs prises par l'argument et l'on aura affaire effectivement à une fonction de

*nom de variable réelle*². Si *exp num* ne fait pas intervenir dans sa définition l'argument, la fonction définie est une fonction constante de l'argument. L'argument est une variable muette et peut être utilisée ailleurs dans le programme sans interférence avec la définition de la fonction.

- une fonction peut faire intervenir dans sa définition n'importe quelle variable, mais le calcul de la fonction ne fournira, toutes choses égales par ailleurs, des résultats différents que si l'expression de calcul fait intervenir effectivement l'argument et si ce dernier varie.

- une fonction peut être définie plusieurs fois dans le cours d'un programme. L'interpréteur ne considérera que la dernière définition fournie.

- *nom de variable*¹ suit les règles de formation et d'identification des noms, c'est à dire en particulier que FN ABC et FN ABD représentent la même fonction.

- la définition de la fonction doit impérativement intervenir avant la première utilisation de cette fonction. La définition est achevée quand l'interpréteur a terminé d'analyser l'instruction sur laquelle elle se trouve : ceci interdit d'utiliser une fonction qui se définisse par elle-même. Dans le cas où l'on tente d'utiliser une fonction sans l'avoir auparavant déclarée il y apparition du message ?UNDEF'D FUNCTION ERROR.

Dans le cas où une fonction intervient dans sa propre définition, ce même message peut apparaître mais d'autres phénomènes peuvent se produire, dépendant alors du contexte.

- l'instruction DEF ne peut être utilisée qu'en différé.

Exemples :

```

]100 DEF FN A(W) = 2 * W + W
]200 PRINT FN A(23)
]300 DEF FN B(X) = 4 + 3
]400 G = FN B ( 23 )
]500 PRINT G
]600 DEF FN A(Y) = FN B(Z) + Y
]700 PRINT FN A ( G )
]800 END
]RUN

```

69

7

14

$FN A(23) = 2 \times 23 + 23 = 69$

$FN B(\text{nombre quelconque}) = 7$

$FN A(7) = 7 + 7 = 14$

5 INSTRUCTIONS D'ASSIGNATION DE VALEUR.

Ce sont : pour les variables définies dans un programme BASIC, l'instruction LET ... =

pour les emplacements mémoire extérieurs au programme BASIC, l'instruction POKE.

5.1 LET

Objet : assigner une valeur à une variable précédemment définie ou non.

Syntaxe : (LET) *nom de variable*
nom de variable (indices) = *expression*

Commentaires : ● le mot LET est facultatif.

- l'expression située à droite du signe = doit fournir un résultat du même type que la variable de nom *nom de variable*.

Si le dernier caractère de *nom de variable* est S, donc la variable est de type alphanumérique et l'expression doit fournir un résultat de type alphanumérique.

Si le dernier caractère de *nom de variable* n'est pas S, alors la variable est numérique et le résultat de l'expression doit être de type numérique.

Si ces contraintes ne sont pas respectées, il y apparition du message ?TYPE MISMATCH ERROR.

- si *expression* est une chaîne de caractères non délimitée par des cotes "", l'interpréteur BASIC APPLESOFT tentera d'interpréter cette chaîne de caractères comme une expression numérique. Si l'expression ainsi interprétée est incorrecte, il y a apparition du message ?SYNTAX ERROR. Si par contre l'expression interprétée a un sens et que *nom de variable* représente une variable alphanumérique, il y affichage du message ?TYPE MISMATCH ERROR.

Exemples :

```

LET A = 1
A2 = 123.2
A$ (2) = "TEST "

```

5.2 POKE

Objet : assigner à une adresse en mémoire une valeur numérique.

Syntaxe : POKE *exp num 1*, *exp num 2*

Commentaires : ● l'instruction POKE place en mémoire à l'adresse résultant de l'évaluation de *exp num 1* la valeur résultant de l'évaluation de *exp num 2*.

- *exp num 1* doit donc fournir une valeur comprise entre -65 535 et 65 535. Les adresses négatives sont identiques aux adresses positives obtenues en leur additionnant 65 536. Ainsi, - 936 et 64 600 se réfèrent à la même adresse.

- *exp num 2* doit fournir une valeur comprise entre 0 et 255, puisque le nombre qui peut être stocké dans un octet est limité à ces valeurs.

- si l'un ou l'autre des deux paramètres sort du domaine autorisé, il y affichage du message ?ILLEGAL QUANTITY ERROR.

- dans les deux expressions permettant d'évaluer les paramètres, les nombres réels sont convertis en nombre entiers juste inférieurs (voir cependant la remarque en 2.5.3).

- l'adresse calculée par l'intermédiaire de *exp num 1* doit appartenir d'abord à l'espace adressable de la machine, ensuite coorespondre à une adresse en mémoire vive (ou RAM) ou bien à une adresse de dispositif de sortie d'informations. Si cette condition n'est pas réalisée, aucune action ne sera exécutée par l'ordre POKE.

Pour une machine équipée de 16 K de mémoire, l'adresse en mémoire vive la plus élevée est 16 383 ; pour une machine de 24 K, elle est de 32767 et pour une machine de 48 K, elle se monte à 49 151. Plus généralement, pour une machine possédant *n* K de mémoire, l'adresse la plus élevée est de $(1\ 024 \times n) - 1$.

Quelle que soit la configuration de la mémoire, les adresses supérieures à 52 348 sont réservées à la mémoire morte donc non modifiables par un ordre POKE.

Les adresses comprises entre 49 152 et 52 347 correspondent aux canaux d'entrée/sortie et donc, suivant les dispositifs connectés, l'ordre POKE aura un effet ou non. Pour cela, il faut se rapporter aux manuels d'utilisation des périphériques connectés.

- une liste non exhaustive de POKE avec leurs paramètres et leur action est donnée en Annexe.
- il faut attirer l'attention de l'utilisateur sur le fait que BASIC APPLESOFT ne contrôle pas la validité fonctionnelle de l'ordre POKE et par conséquent l'utilisateur peut altérer le contenu d'une adresse quelconque en mémoire vive. Si cette adresse contenait des informations nécessaires au bon fonctionnement (du point de vue logique) de l'appareil, il peut résulter du POKE des incidents de fonctionnement allant jusqu'à la perte du programme en cours.

Exemples :

POKE 33,30 inscrit à l'adresse décimale 33 la valeur 30
POKE - 16 303, SIN (0) inscrit à l'adresse décimale 65 536 - 16 303
soit 49 233, le nombre entier résultant du calcul SIN (0) soit 0.

6 INSTRUCTIONS DE CONTROLE DU DEROULEMENT D'UN PROGRAMME.

Les instructions d'un programme BASIC APPLESOFT s'exécutent normalement en séquence, c'est à dire que, lorsqu'une instruction est achevée, on exécute la suivante. Pour altérer la séquence des instructions, on a recours à diverses instructions qui permettent d'indiquer où aller dans le programme. Ce "débranchement" dans la suite physique des instructions peut être conditionnel ou obligatoire.

On peut aussi modifier la séquence des instructions pour opérer des traitements répétitifs : on dispose ainsi d'instructions qui permettent de revenir "sur ses pas" pour recommencer les traitements et ceci jusqu'à la rencontre d'une condition de fin de "boucle".

Pour simplifier l'écriture d'un programme, on dispose d'instructions qui permettent d'exécuter un même groupe d'instructions plusieurs dans le cours du programme sans se "débrancher" explicitement : on dit qu'on fait appel à un sous-programme. Ces sous-programmes peuvent être écrits en BASIC APPLESOFTS ou bien en langage machine.

Il existe une catégorie d'instruction permettant de ne pas interrompre le déroulement d'un programme en cas d'erreur et de prendre les mesures nécessaires pour les corriger par programme.

Une dernière catégorie d'instructions permet de mettre fin à l'exécution d'un programme ou bien de le faire attendre pendant un temps déterminé.

6.1 GOTO

Objet : aller exécuter les instructions à partir d'une ligne donnée en séquence.

Syntaxe : GOTO numéro de ligne

Commentaires : ● le numéro de la ligne au début de laquelle on désire se brancher doit être explicité, et non calculé.

● si numéro de ligne est absent ou correspond à une ligne inexistante dans l'énoncé du programme, il y affichage du message ?UNDEF'D STATEMENT ERROR IN numéro de la ligne comportant le GOTO erroné.

● si numéro de ligne n'est pas numérique, il y a affichage du message ?SYNTAX ERROR.

Exemples :

]10 GOTO 30 aller exécuter les instructions à partir de 30
]
]15 PRINT " 15 " imprimer 15

]20 END fin du programme
]
]30 PRINT " 30 " imprimer 30
]
]40 GOTO 15 aller en 15
]
]RUN exécution du programme ci-dessus
]
30)
) résultats
15)

6.2 ON ... GOTO

Objet : aller exécuter les instructions à partir d'un numéro de ligne pris dans une liste par le calcul du rang dans cette liste.

Syntaxe : ON exp num GOTO n° ligne 1, (n° ligne 2), ..., (n° ligne n)

Commentaires : ● si exp num est positif et inférieur à 256, il est converti si nécessaire en nombre entier (notons ce dernier k) et le programme va s'exécuter à partir de la ligne dont le numéro est le k^{ième} dans la liste, soit n° ligne k.

● si k vaut 0 ou est plus grand que le nombre d'éléments de la liste de numéros de ligne, le programme continue en séquence, c'est à dire que le GOTO est sans effet.

● si l'élément de la liste choisi est un numéro de ligne inexistant dans le programme, il y a affichage du message ?UNDEF'D STATEMENT ERROR IN numéro de la ligne contenant l'instruction ON ... GOTO

● si exp num est évaluée à un nombre négatif ou supérieur à 255, il y a affichage du message ?ILLEGAL QUANTITY ERROR.

Exemples :

]10 I = I + 1 initialement I = 0
]
]20 ON I GOTO 40,50 si I = 1, aller en 40 ; si I = 2
 aller en 50 ; sinon aller en
]
]30 END 30 : fin du programme
]
]40 PRINT " * ")
) imprimer une étoile puis revenir en 10
]45 GOTO 10)
]
]50 PRINT " ** ")
) imprimer deux étoiles puis revenir en 10
]60 GOTO 10)
]
]RUN exécuter le programme ci-dessus
*)
) résultats
**)

IF ... THEN ...

Objet : exécuter les instructions qui suivent de façon conditionnelle : si la condition est vraie, on exécute ce qui suit THEN.

Syntaxe : IF *exp log* THEN *instructions*
numéro de ligne

Commentaires : ● si *exp log* est vraie, on exécute la ou les instructions qui suivent THEN sur la ligne. Dans le cas contraire, on passe à l'exécution de la ligne numérotée suivante. Si THEN est suivi de *numéro de ligne*, l'exécution continue à la ligne indiquée si *exp log* est vraie ; sinon, l'exécution continue en séquence.

● si *exp log* est vraie et *numéro de ligne* est une ligne inexistante dans le programme, il y a affichage du message ?UNDEF'D STATEMENT ERROR IN *numéro de la ligne contenant l'instruction IF ... THEN*

● si *exp log* est en fait une chaîne de caractères, et donc est fausse par nature, plus de deux occurrences risquent de causer une erreur et il y a affichage du message ?FORMULA TOO COMPLEX ERROR. Néanmoins un telle condition a peu d'intérêt et donc est à éviter.

● si le champ qui suit THEN est vide, aucune n'action n'est effectuée et l'exécution continue en séquence.

● la description des expressions logiques est faite au 2.7.2 et suivants.

● THEN ne peut être utilisé seul : il résulte de son emploi non précède de IF le message ?SYNTAX ERROR.

Exemples :

]10 IF NOT I THEN 30	<i>si I vaut 0, aller en 30 ;</i>
	<i>sinon en séquence, donc en 20.</i>
]20 IF I = 2 THEN END	<i>Si I vaut 2, fin du programme ;</i>
	<i>sinon en séquence donc 30 :</i>
]30 PRINT I	<i>imprimer I</i>
]40 I = I + 1	<i>incréméntation de I</i>
]50 GOTO 10	<i>retour en 10</i>
]RUN	<i>exécution du programme ci-dessus</i>
0	)
	) résultats
1	)

IF ... GOTO

Objet : aller exécuter les instructions à partir d'une ligne donnée si une condition est réalisée.

Syntaxe : IF *exp log* GOTO *numéro de ligne*

Commentaires : ● cette instruction est identique à l'instruction IF *exp log* THEN *numéro de ligne*. On se reportera donc au 6.3 pour les règles d'utilisation.

Exemples :

]10 IF I GOTO 40	<i>initialement I = 0</i>
	<i>si I est différent de 0,</i>
	<i>aller en 40.</i>
]20 I = I + 1	<i>I vaut maintenant 1</i>
]30 GOTO 10	<i>aller en 10</i>
]40 END	<i>fin du programme</i>

FOR

Objet : permettre la répétition de l'exécution d'un groupe d'instructions situées entre FOR et NEXT. Le nombre de répétitions est fonction des valeur valeur initiale et finale de la variable de boucle et du pas éventuel.

Syntaxe : FOR *variable réelle* = *exp num 1* TO *exp num 2* (STEP *exp num 3*)

Commentaires : ● la variable réelle de nom *variable réelle* reçoit la valeur à laquelle est évaluée *exp num 1* et les instructions sont exécutées jusqu'à la rencontre d'une instruction NEXT ou NEXT *variable réelle*. *Variable réelle* est alors incrémentée de la valeur à laquelle est évaluée *exp num 3*. Si le paramètre de pas de la boucle est absent, c'est à dire en l'absence de l'option STEP, le pas vaut par défaut 1. On compare cette nouvelle valeur de *variable réelle* à la valeur à laquelle est évaluée *exp num 2*. Si *variable réelle* est strictement supérieure à la borne supérieure, l'exécution continue en séquence après l'instruction NEXT *variable réelle*. Si *variable réelle* est inférieure à la borne supérieure de la boucle, on exécute de nouveau les instructions qui suivent l'instruction FOR *variable réelle* =

● si la variable de boucle n'est pas une variable réelle, il y a affichage du message ?SYNTAX ERROR.

● les expressions *exp num 1*, *exp num 2* et *exp num 3* sont des expressions numériques quelconques, totalement indépendantes l'une de l'autre.

● si le pas est négatif, c'est à dire si prend une valeur négative, le fonctionnement de la boucle FOR est inversé.

La condition de fin de boucle est que *variable réelle* soit strictement inférieure à la valeur prise par *exp num 2*. Dans le cas contraire, l'exécution reprend à l'instruction suivant l'instruction FOR *variable réelle* =

● le test de fin de boucle est effectué à la fin de la boucle, ce qui signifie que même si la borne supérieure est inférieure à la borne inférieure, avec un pas positif, la boucle sera exécutée quand même une fois.

● les boucles FOR ... NEXT ne doivent pas se croiser. Si tel est le cas, il y a affichage du message ?NEXT WITHOUT FOR ERROR.

<i>CORRECT</i>	<i>INCORRECT</i>
1 FOR I = ...	1 FOR I = ...
2 FOR J = ...	2 FOR J = ...
3 NEXT J	3 NEXT I
4 NEXT I	4 NEXT J

Dans le premier exemple, on attend que la boucle intérieure soit achevée pour passer dans la boucle extérieure. Par contre, dans le deuxième cas, on recommence chaque fois la boucle intérieure sans passer sur une instruction NEXT. Pour éviter ceci, un message d'erreur est affiché.

● on ne doit pas imbriquer plus de 10 boucles FOR ... NEXT. Si cette limite est dépassée, il y a affichage du message ?OUT OF MEMORY ERROR.

● on ne doit pas sortir de la boucle FOR autrement que par le test de fin de boucle. Si par exemple on sort de la boucle par un GOTO, et que dans le cours du programme on laisse ainsi plus de 10 boucles FOR ... NEXT actives, on aura le message ?OUT OF MEMORY ERROR.

● si deux boucles ont la même variable, c'est à dire que les deux premiers caractères de la variable sont identiques, la boucle la plus ancienne est rendue inactive, c'est à dire que l'on est dans même situation que si l'on était sorti de la boucle de la manière normale.

● en mode d'exécution immédiat, les instructions FOR et NEXT doivent se trouver sur la même ligne pour obtenir un fonctionnement correct de la boucle. Une ligne contient, on le rappelle jusqu'à 239 caractères. Pour

plus de détails sur le fonctionnement de la boucle FOR ... NEXT en mode immédiat, on se reportera à l'instruction NEXT en 6.6.

● une boucle FOR ... NEXT occupe 16 octets pour ses pointeurs quand elle est active, c'est à dire qu'elle n'a pas été achevée par le test de fin de boucle.

Exemples :

```
]10 FOR I=1 TO 5 STEP 3  on sort de la boucle si
                        I est supérieur à 5.
]20 FOR J=3 TO 4         Les valeurs prises par I sont 1 et 4.
                        On sort de la boucle intérieure
]30 PRINT " I=" ; I ; " J=" ; J  Jsi J est supérieur à 4.
                        J prend les valeurs 3 et 4.
]40 NEXT J              I=I+1
                        si I supérieur à 5, fin du programme ;
]50 NEXT I              sinon aller en 10
```

]RUN

```
I=1 J=3      )
I=1 J=4      ) résultats
I=4 J=3      )
I=4 J=4      )
```

```
]FOR I=2 TO 4 : PRINT I I: programme en mode immédiat
NEXT          imprimant I puissance I
              pour I compris entre 2 et 4.
4             L'instruction NEXT est sur
27            la même ligne logique que FOR.
256           Les : sont les séparateurs d'instruction.
```

6.6 NEXT

Objet : indiquer la fin d'un groupe d'instructions inclus dans une boucle initialisée par FOR.

Syntaxe : NEXT variable de boucle 1 (, variable de boucle 2 (, ...))

Commentaires : ● les noms de variables situés derrière NEXT et séparés par des virgules sont des noms de variables de boucles FOR. C'est à dire que pour chacune des variables variable de boucle 1, variable de boucle 2, ..., il existe dans le programme des ordres FOR variable de boucle 1, FOR variable de boucle 2, ..., qui ont été déjà exécutés. Si NEXT n'est suivi d'aucun nom de variable, il constitue l'instruction NEXT de la dernière boucle FOR exécutée.

● l'ordre dans lequel sont énumérées les variables de boucle doit correspondre à l'ordre inverse d'imbrication des instructions FOR correspondantes. Dans le cas contraire, il y a affichage du message ?NEXT WITHOUT FOR ERROR.
En effet, l'instruction NEXT variable de boucle 1, variable de boucle 2,... est identique à la suite d'instructions NEXT variable de boucle 1, NEXT variable de boucle 2,...

● si aucune variable n'est spécifiée après NEXT et qu'aucune boucle FOR n'est active, il y a affichage du message ?NEXT WITHOUT FOR ERROR.

● si l'une au moins des variables citées dans la liste suivant NEXT n'est pas une variable de boucle, c'est à dire pour laquelle il n'existe pas d'instruction FOR correspondante, il y a affichage du message ?NEXT WITHOUT FOR ERROR.

● si un nom de variable de la liste suivant NEXT n'est pas un nom de variable réelle, il y a affichage du message ?SYNTAX ERROR.

● l'instruction NEXT sans liste de variable s'exécute plus rapidement que la même instruction suivie d'une liste de variable. Cela peut néanmoins rendre fastidieuse la compréhension des programmes.

● si une boucle FOR ... NEXT est encore active malgré l'arrêt d'un programme (ce qui peut être le cas quand une erreur apparaît dans le cours d'un programme), une instruction NEXT sans paramètres peut permettre de retourner dans le programme interrompu à l'endroit approprié.

● si une instruction FOR non suivie de NEXT sur la même ligne logique (une ligne logique contient jusqu'à 239 caractères), a été rentrée en mode immédiat, l'exécution en mode immédiat de NEXT cause en général l'apparition du message ?SYNTAX ERROR, sauf si NEXT est sans paramètres et si aucune instruction n'a été entrée depuis l'exécution de la boucle FOR : dans ces conditions, NEXT fonctionne comme si il était situé sur la ligne contenant FOR.

```
FOR I=1 TO 5 : PRINT I  ligne d'instructions
1                      résultat
NEXT                   instruction au clavier
2                      résultat
NEXT                   instruction au clavier
3                      résultat
NEXT I                 instruction au clavier
?SYNTAX ERROR IN 65738 message d'erreur ; le
                       numéro de ligne indiqué
                       est sans signification.
```

Exemple :

```
]10 FOR I=1 TO 2
]20 FOR J=1 TO 2
]30 FOR K=1 TO 2
]40 NEXT
]50 NEXT J, I          NEXT se rapporte à la
                       dernière boucle activée
```

6.7 SOUS-PROGRAMMES.

Il est souvent nécessaire d'utiliser la même procédure dans le cours d'un programme. Plutôt que de récrire cette procédure chaque fois qu'on en a besoin, il est possible de faire appel à une série d'instructions qui effectuent la procédure désirée, cette série d'instruction étant écrite une fois pour toutes. Cette suite d'instructions est appelée sous-programme. Elle est exécutée par GOSUB.

Il peut être aussi nécessaire d'utiliser des programmes écrits en langage machine, qui ont des fonctions que le BASIC APPLESOFT ne peut pas fournir, ou dont les temps d'exécution sont très inférieurs à ceux d'un programme BASIC effectuant les mêmes traitements. Ces programmes en langage machine sont appelés par l'instruction CALL.

6.7.1 GOSUB

Objet : exécuter les instructions situées entre le numéro de ligne indiqué et l'instruction RETURN, puis continuer en séquence.

Syntaxe : GOSUB numéro de ligne

Commentaires : ● l'instruction GOSUB a pour effet de conserver l'adresse de l'instruction suivant GOSUB et d'effectuer un GOTO numéro de ligne.

● en conséquence, si le numéro de ligne indiqué est inexistant dans le programme, il y a affichage du message ?UNDEF'D STATEMENT IN numéro de la ligne contenant l'instruction GOSUB erronée.

● l'adresse de retour du sous-programme, c'est à dire l'adresse de l'instruction GOSUB, est stockée dans une pile dont la taille ne permet pas de contenir plus de 25 adresses. Il est donc impossible d'appeler plus de 25 sous-programmes en même temps. Il est de la même façon interdit à un sous-programme de s'appeler lui-même si cela conduit à dépasser le nombre de 25 appels de sous-programmes au total. Si cette limite est dépassée, il y a dans tous les cas affichage du message ?OUT OF MEMORY ERROR.

● il est de la même façon fortement déconseillé de sortir d'un sous programme autrement que par les instructions appropriées RETURN et POP décrites plus loin, car ce sont les seules instructions qui permettent de libérer l'adresse stockée dans la pile.

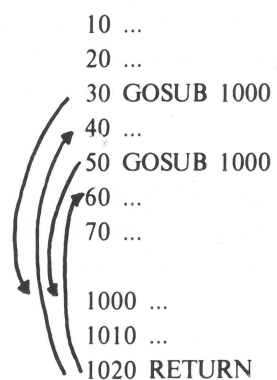
● un sous-programme actif réserve 6 octets pour ses pointeurs. Un sous-programme est actif tant qu'il ne rencontre pas l'instruction RETURN ou POP.

Exemples :

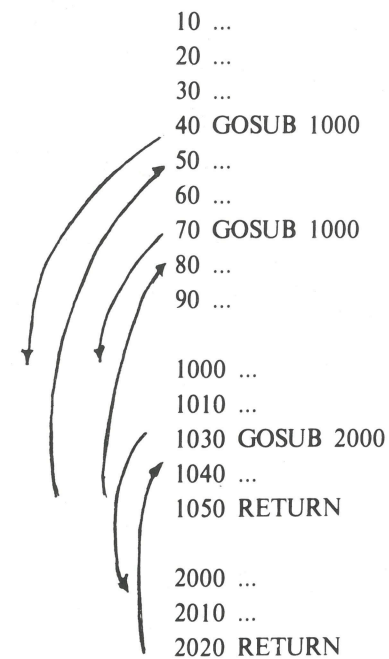
]10 GOSUB 40	<i>aller en 40 et après l'instruction RETURN continuer en séquence donc en 20</i>
]20 PRINT "RETOUR "	
]30 END	<i>fin du programme</i>
]40 PRINT "SOUS PROGRAMME "	<i>début du sous-programme aller en l'adresse qui suit le dernier ordre GOSUB</i>
]50 RETURN	<i>exécuter le programme ci-dessus</i>
]RUN	

SOUS PROGRAMME	)
	) résultats
RETOUR	)

Il est possible d'appeler un sous-programme à partir de plusieurs endroits d'un programme. Le déroulement du programme apparaît alors le suivant :



On peut affiner l'illustration du déroulement d'un programme lorsqu'un sous-programme appelle lui-même un autre sous-programme :



6.7.2 ON ... GOSUB

Objet : ● appeler un sous-programme dont le numéro de ligne est pris dans une liste à partir du calcul du rang dans cette liste.

Syntaxe : ON exp num GOSUB n° ligne 1 (, n° ligne 2 (...))

Commentaires : ● ON ... GOSUB a pour effet de conserver l'adresse de l'instruction qui le suit et de se comporter comme ON exp num GOTO n° ligne 1 (, n° ligne 2 (...)).

● cet appel de sous programme paramètre par l'expression exp num suit donc les mêmes règles de syntaxe que ON ... GOTO ... auquel on se reportera avec intérêt en 6.2.

● fonctionnellement, il se comporte comme un GOSUB et c'est à ce chapitre qu'on trouvera tous les commentaires nécessaires (en 6.7.1).

Exemple :

10 FOR I=1 TO 2	
20 ON I GOSUB 50, 70	<i>si I vaut 1 exécuter le sous programme à partir de 50</i>
30 NEXT I	<i>si I vaut 2, exécuter le sous programme à partir de 70</i>
40 END	
50 PRINT "SOUS PROGRAMME 1 "	
60 RETURN	<i>fin du sous programme ; retour en 30</i>
70 PRINT "SOUS PROGRAMME 2 "	

80 RETURN	<i>fin du sous programme ;</i>
	<i>retour en 30</i>
RUN	<i>exécution du programme</i>
SOUS PROGRAMME 1	)
	<i>) résultats</i>
SOUS PROGRAMME 2	)

6.7.3 RETURN

Objet : indiquer la fin d'un sous-programme : donner l'ordre de retour dans le programme qui a appelé le sous programme.

Syntaxe : RETURN

Commentaires : ● RETURN s'emploie tout seul. Il indique qu'on quitte l'ordre séquentiel des instructions pour aller exécuter les instructions en séquence à partir de l'adresse stockée par l'instruction GOSUB antérieure.

● si aucune instruction GOSUB n'a été précédemment exécutée, il y a affichage du message ?RETURN WITHOUT GOSUB ERROR. Ceci est le cas en particulier quand on se branche dans un sous-programme par un ordre GOTO au lieu de GOSUB.

● on peut mettre plusieurs RETURN dans un sous-programme mais c'est le premier rencontré qui c'est le premier rencontré qui provoquera le retour dans le programme appelant.

● le message ?RETURN WITHOUT GOSUB ERROR apparait aussi quand il a été exécuté plus d'instructions RETURN que d'ordres GOSUB correspondants.

Exemples :

10 FOR I=1 TO 10	<i>ce programme appelle 10 fois le sous-programme</i>
20 GOSUB 100	<i>situé à la ligne 100 et suivantes.</i>
30 NEXT I	
40 END	
100 IF I= < 9 THEN RETURN	<i>si I est inférieur ou égal à 9, retour en 30</i>
110 PRINT I	<i>sinon imprimer I</i>
120 RETURN	<i>puis retour en 30</i>
RUN	<i>exécution du programme</i>
10	<i>résultat</i>

6.7.4 POP

Objet : permettre de simuler une instruction RETURN sans opérer le branchement à la dernière stockée par le dernier ordre GOSUB rencontré.

Syntaxe : POP

Commentaires : ● POP s'emploie sans paramètres.

● POP a pour objet de remonter d'une place dans la pile des adresses de retour de sous-programmes. Ainsi, après une instruction POP, la prochaine instruction RETURN n'opérera pas un retour à l'instruction suivant le dernier GOSUB, mais celle suivant l'avant dernier GOSUB rencontré.

● l'instruction POP agissant en partie comme l'instruction RETURN, elle obéit aux mêmes règles. Si l'on rencontre plus d'instructions POP ou RETURN qu'on n'a rencontré d'instructions GOSUB, il y a affichage du message ?RETURN WITHOUT GOSUB ERROR.

Exemple :

10 GOSUB 100	<i>appel du sous-programme commençant en 100, puis</i>
20 END	<i>fin du programme.</i>
100 GOSUB 200	
110 PRINT " 110 "	<i>ce sous programme appelle le sous programme en 200,</i>
120 RETURN	<i>imprime 110 et retour</i>
200 POP	
210 RETURN	<i>ce sous programme exécute un POP ; l'adresse de retour est donc celle du sous</i>
RUN	<i>programme précédent ; dans le cas présent : 20</i>
	<i>En conséquence, l'instruction 110 est sautée et rien n'apparait !.</i>

6.7.5 CALL

Objet : permettre d'exécuter un sous programme écrit en langage machine et, si ce dernier le permet, reprendre l'exécution en séquence.

Syntaxe : CALL exp num

Commentaires : ● exp num doit prendre une valeur compris entre - 65 535 et 65 535. Elle représente l'adresse décimale à laquelle on désire se brancher. Si l'adresse est négative, elle correspond à l'adresse positive obtenue en ajoutant 65 536. Si exp num prend une valeur extérieure au domaine indiqué, il y a affichage du message ?ILLEGAL QUANTITY ERROR.

● en pratique, l'adresse obtenue à partir de exp num doit appartenir soit au domaine physiquement adressable de mémoire vive soit au domaine physiquement adressable de mémoire morte. Le domaine de mémoire vive physiquement adressable dépend de la taille mémoire de la configuration : si la machine possède n K octets de mémoire vive, le domaine adressable s'étend de 0 à (1024 × n) - 1. Le domaine adressable en mémoire morte dépend lui-aussi de la configuration et des circuits de mémoire morte qui sont installés : néanmoins, le domaine adressable physiquement est toujours compris entre 49 152 et 65 535, ou, en nombre négatifs, entre - 1 et - 16 384.

● le programme ne rendra le contrôle au programme écrit en BASIC APPLESOFT que s'il dispose pour se terminer d'une instruction de retour notée en langage machine RTS. Il est donc de la responsabilité du programmeur de s'assurer que le programme en langage machine permet de revenir " sous " BASIC.

● comme on l'a déjà remarqué pour l'instruction POKE au 5.2, l'interpréteur BASIC APPLESOFT n'est pas en mesure de contrôler la validité fonctionnelle d'un programme qui ne s'effectue pas sous son contrôle et il est de la responsabilité du programmeur de ne pas exécuter des instructions qui peuvent mettre en péril le contenu de certaines zones en mémoire réservées au BASIC APPLESOFT. Dans le cas où ces adresses seraient altérées, il y a une forte probabilité pour qu'on ne puisse par revenir au programme BASIC appelant, ce qui signifie par exemple la perte du programme en mémoire.

- un certain nombre de sous-programmes, dont le fonctionnement est sûr, se trouvent dans les mémoires mortes du Moniteur, et permettent d'accéder à des fonctions que BASIC ne fournit pas :

CALL - 958 permet par exemple d'effacer l'écran à partir de la position courante du curseur jusqu'à la dernière ligne.

- une liste, non exhaustive, de ces adresses est donnée en Annexe.

6.8 ON ERR GOTO

Objet : permettre à l'utilisateur de traiter lui même les erreurs détectées par l'interpréteur BASIC APPLESOFT.

Syntaxe : ONERR GOTO *numéro de ligne*

Commentaires : ● cette instruction n'a pas d'effet immédiat : elle signale seulement à l'interpréteur BASIC APPLESOFT qu'il existe une adresse où se brancher en cas d'erreur qui est différente de l'adresse standard. Ceci est nécessaire si par exemple on veut éviter de voir le programme s'interrompre si une faute de frappe est commise lors de l'entrée de données, ou bien si l'on ne désire pas voir apparaître de message en Anglais.

- *numéro de ligne* doit exister si l'on ne veut pas voir s'afficher le message ?UNDEF'D STATEMENT ERROR IN *numéro de la ligne comportant l'ordre ONERR ... GOTO*.

- en cas d'erreur, le contrôle est donné aux instructions situées sur la ligne indiquée. Un code erreur peut être trouvé à l'adresse *décimale* 222. Il peut donc être obtenu par l'instruction PEEK (222). Cette adresse contient non seulement le code d'erreurs détectées par l'interpréteur BASIC APPLESOFT, mais aussi, dans le cas d'utilisation du système disque DOS, le code des erreurs détectées par le système. On ne donne ici que la liste des codes correspondant aux erreurs du BASIC APPLESOFT, mais une liste complète est donnée en Annexe.

- le numéro de la ligne sur laquelle s'est produit l'erreur qui a provoqué le branchement sur les instructions de traitement des erreurs peut être obtenu par la formule : PEEK (218) + PEEK (219)*256.

- la liste ci-dessous contient le code, le libellé standard et une traduction du code erreur pour les erreurs propres au BASIC APPLESOFT.

CODE	LIBELLE STANDARD	TRADUCTION
0	NEXT WITHOUT FOR ERROR	NEXT sans instruction FOR correspondante.
16	SYNTAX ERROR	Erreur de syntaxe.
22	RETURN WITHOUT GOSUB ERROR	RETURN sans instruction GOSUB correspondante.
42	OUT OF DATA ERROR	Fin des données dans l'instruction DATA (mais intervient lors d'une instruction READ).
53	ILLEGAL QUANTITY ERROR	Quantité illégale.
69	OVERFLOW ERROR	Dépassement de capacité.
77	OUT OF MEMORY ERROR	La capacité de la pile de stockage des adresses de boucles ou de sous-programmes est dépassée.
90	UNDEF'D STATEMENT ERROR	Numéro de ligne inconnu.
107	BAD SUBSCRIPT ERROR	Un indice dépasse la valeur maximum assignée par l'ordre DIM ou la valeur 10 si le tableau n'a pas été dimensionné.

120	REDIM'D ARRAY ERROR	Un tableau a été l'objet de deux instructions DIM ou cette dernière est intervenue après la première utilisation du nom du tableau.
133	DIVISION BY ZERO ERROR	Division par zéro ou un nombre inférieur à 2.8 E - 39.
163	TYPE MISMATCH ERROR	Les variables à droite et à gauche d'un signe = sont de types incompatibles.
176	STRING TOO LONG ERROR	Une chaîne de caractères dépasse 255 caractères de long.
191	FORMULA TOO COMPLEX ERROR	Plus de deux ou trois conditions IF avec des chaînes de caractères comme expressions logiques.
224	UNDEF'D FUNCTION ERROR	Une fonction est utilisée sans avoir été définie par un ordre DEF FN.
254	BAD RESPONSE TO INPUT STATEMENT	Réponse incorrecte à une instruction INPUT.
255	BREAK IN <i>numéro de ligne</i>	Interruption du programme demandée par l'opérateur au moyen de CTRL C.

- pour désactiver le branchement en cas d'erreur sur le programme utilisateur et pour revenir donc au traitement standard, il suffit de mettre à zéro l'adresse *décimale* 216 par l'intermédiaire de l'instruction POKE 216.0.

- un certain nombre de problèmes peuvent apparaître lors du traitement des erreurs par l'utilisateur dans les cas suivants :

- l'erreur intervient à l'intérieur d'une boucle FOR ... NEXT
- l'erreur intervient à l'intérieur d'un sous programme BASIC
- l'erreur intervient sur une instruction GET
- le programme est exécuté en mode TRACE ou contient une instruction PRINT (ce qui est le cas de la grande majorité des programmes).

L'effet de l'instruction ONERR ... GOTO sur le comportement des programmes dans les cas cités ci-dessus est discuté plus bas. Néanmoins, il existe un moyen simple d'éviter des incidents de fonctionnement en appelant un sous-programme en langage machine dans les instructions de traitement d'erreur. Ce sous programme peut être soit rentré à la main en langage machine à partir du moniteur, soit installé en mémoire par l'intermédiaires de POKE, soit enfin chargé à partir du disque si on en dispose. On peut installer ce sous-programme à l'adresse 768 et suivantes. On aura alors à faire un CALL 768 parmi les instructions de traitement d'erreurs.

Dans le cas où le sous programme est installé en 768 à 777 *décimal* soit 300 en hexadécimal, on l'installera comme suit :

- à partir du Moniteur :
 - appuyer sur la touche RESET ou taper CALL - 151 pour quitter le BASIC APPLESOFT ;
 - taper : 300 : 68 A8 68 A6 DF 9A 48 98 48 60 *suivi de RETURN* ;
 - taper *ctrl C* suivi de *RETURN* pour revenir sous BASIC APPLESOFT.
- à partir d'un programme BASIC APPLESOFT :
 - il suffit d'installer l'équivalent décimal du code hexadécimal cité plus haut avant tout appel du sous programme :
POKE 768, 104 : POKE 769, 168 : POKE 770, 104 : POKE 771, 166 : POKE 772, 223 :
POKE 773, 154 : POKE 774, 72 : POKE 775, 152 : POKE 776, 72 : POKE 777, 96

- à partir du disque :

- il suffit de charger la zone 768 - 777 *décimaux* par l'un des deux moyens cités ci-dessus, puis de charger cette zone sur disque avec l'ordre suivant BSAVE ERREUR, A768, L10 puis, dans chaque programme utilisant ce programme de traitement d'erreur, de faire : 10 PRINT " *ctrl* D BLOAD ERREUR, A768 "

- ERREUR est un nom de fichier donc pris à des fins d'exemple.

● dans le cas où l'erreur se serait produite dans une boucle FOR ... NEXT, les pointeurs gérant la boucle étant perdus, il faut envisager de faire repartir la boucle à partir d'un ordre FOR. En effet, la boucle est devenue inactive et la rencontre du premier NEXT causerait une erreur du type NEXT WITHOUT FOR ERREUR.

● dans le cas où l'erreur se serait produite dans un sous-programme BASIC, l'adresse de retour est perdue et la rencontre du premier RETURN ou POP cause l'apparition d'une erreur du type RETURN WITHOUT GOSUB ERROR. Il faut donc envisager de relancer l'exécution du sous programme où s'est produit l'incident à partir du programme ou sous programme appelant.

● si l'instruction ONERR GOTO est utilisée conjointement avec l'instruction RESUME pour traiter les erreurs survenant dans une instruction GET, le programme se suspend indéfiniment si deux instructions GET **consécutives** tombent en erreur. Néanmoins, si le programme de traitement d'erreur se termine par un GOTO, le fonctionnement reste correct (mais pas dans le cas exposé ci-après).

● Quand il est utilisé dans un programme exécuté en mode TRACE ou contenant uns instruction PRINT, l'ordre ONERR GOTO cause un abandon du programme et un branchement sur le Moniteur après la 43^{ième} erreur. Quand les erreurs proviennent d'un ordre INPUT, le cas est légèrement différent :

si l'ordre RESUME est utilisé pour reprendre l'exécution, le fonctionnement reste coorect ;

si l'ordre GOTO est utilisé pour reprendre l'exécution normale, il y a abandon du programme et branchement sur le Moniteur après la 87^{ième} erreur.

● dans tous tous les cas où il y aurait branchement sous le Moniteur, la séquence RESET *ctrl* C permet de revenir sous BASIC APPLESOFT.

Exemple :	0 ONERR GOTO 1000	<i>indique que le traitement des erreurs se fait par</i>
	10 ...	<i>LES INSTRUCTIONS EN 1000</i>
	20 ...	<i>et suivantes.</i>
	1000 Y = PEEK (222)	<i>assigne à Y la valeur du code erreur.</i>
	1010 PRINT " ERREUR NUMERO " ; Y	<i>Cette instruction imprime un message contenant le code erreur</i>
	1020 END	<i>puis fin du programme.</i>

Si par exemple on avait dans ce programme une erreur due à une division par zéro, le message suivant serait imprimé : ERREUR NUMERO 133.

6.9 RESUME

Objet : permettre de reprendre l'exécution du programme sur l'instruction qui a causé une erreur.

Syntaxe : RESUME

Commentaires : ● l'instruction RESUME permet d'exécuter de nouveau une instruction qui a causé une erreur. RESUME *doit* se trouver dans le groupe d'instructions qui traite les erreurs. Si RESUME apparait en dehors d'un groupe d'instructions sur lequel on se branche par l'intermédiaire de l'instruction ONERR GOTO, on risque d'exécuter

l'instruction RESUME sans qu'aucune erreur n'ait été rencontrée. Divers phénomènes peuvent apparaitre alors, dépendant du contexte :

- le message ?SYNTAX ERROR IN 65278

- une suspension infinie du programme

- ...

● si l'erreur est située dans le bloc de traitement d'erreur et que RESUME est utilisé pour reprendre l'exécution sans que la cause d'erreur ait disparu, le programme boucle indéfiniment. Pour reprendre le contrôle du programme, utiliser la séquence RESET *ctrl* C.

● si RESUME est utilisé en mode immédiat, la machine risque de se mettre dans un état de suspension infinie ou bien de se brancher sur un programme quelconque en mémoire, qu'il ait été effacé ou non.

Exemple :

10 ONERR GOTO 1000	<i>le traitement d'erreur se fait en 1000 et suivantes.</i>
20 ...	
30 ...	
1000 Y = PEEK (222)	<i>en cas d'erreur, afficher le code erreur et reprendre l'exécution sur l'instruction erronée.</i>
1010 PRINT " ERREUR NUMERO " ; Y	
1020 RESUME	

*Il faut prendre garde au fait suivant : si rien ne vient modifier les instructions en erreur, un tel programme va boucler indéfiniment et seule la séquence RESET *ctrl* C permettra de reprendre " la main ".*

6.10 END

Objet : mettre fin à un programme et rendre le contrôle du calculateur à l'utilisateur.

Syntaxe : END

Commentaires : ● END indique la fin d'un programme. Toutes les commandes BASIC APPLESOFT et instructions sont exécutées en mode immédiat à partir de cette instruction jusqu'à l'introduction d'une instruction RUN.

● END ne modifie pas les valeurs des variables utilisées dans le programme qu'elle achève.

● il peut y avoir plusieurs instructions END dans l'énoncé d'un programme. La première rencontrée met fin à son exécution immédiatement.

● l'instruction END n'est pas nécessaire si elle est physiquement la dernière du programme.

● les pointeurs des boucles et sous programmes restent inchangés.

6.11 STOP

Objet : mettre fin à l'exécution du programme, afficher un message et rendre le contrôle à l'utilisateur.

Syntaxe : STOP

Commentaires : STOP est identique à l'instruction END, à laquelle on se reportera avec intérêt, du point de vue fonctionnel mais affiche un message indiquant le numéro de la ligne où a été rencontré l'instruction STOP.

- le message affiché est le suivant : BREAK IN *numéro de la ligne contenant l'ordre STOP*

Exemple :

10 STOP	<i>l'énoncé du programme est limité</i>
RUN	<i>à l'instruction STOP exécution du programme</i>
BREAK IN 10	<i>message causé par l'ordre STOP</i>

6.12 WAIT

Objet : mettre le programme en attente jusqu'à ce qu'un dispositif externe présente une condition de fin d'attente.

Syntaxe : WAIT *exp num 1, exp num 2 (, exp num 3)*

Commentaires : ● *exp num 1* permet d'évaluer l'adresse *décimale* de l'octet testé. Logiquement compris entre - 65 535 et 65 535 pour éviter le message ?ILLEGAL QUANTITY ERROR, pratiquement compris dans le domaine de mémoire vive adressable est compris entre 0 et (1024*n) - 1 ; et dans cette dernière zone qu'on vient logiquement tester l'état d'un périphérique. Un nombre négatif adresse le même octet que le nombre positif obtenu en lui ajoutant 65 536.

● *exp num 2* permet d'évaluer la valeur *décimale* du masque de huit bits que l'on va appliquer sur l'octet adressé par *exp num 1* pour détecter la condition de fin d'attente. Un bit à 0 dans le masque indique qu'on ne teste pas ce bit dans l'octet testé, un bit à 1 indique le contraire. Le masque ayant 8 bits de long, il doit être évalué par une expression comprise entre 0 et 255. Si la valeur prise par *exp num 2* est extérieure à ce domaine, il y a affichage du message ?ILLEGAL QUANTITY ERROR.

● *exp num 3* est facultative et évaluée à 0 si elle est absente. Elle permet d'évaluer la valeur *décimale* d'un octet dont chaque bit indique si le bit correspondant de l'octet adressé par *exp num 1* doit être inversé (valeur 1 ou laissé tel quel (valeur 0). Ce mot ayant 8 bit de long, il doit être évalué par une expression dont la valeur est comprise entre 0 et 255 pour éviter le message ?ILLEGAL QUANTITY ERROR.

● la condition de fin d'attente est indiquée par la non nullité du *produit logique de exp num 2 avec le OU EXCLUSIF de exp num 3 avec le contenu l'octet adressé par exp num 1.*

Ceci signifie en clair :

- pour chacun des bits de l'octet adressé par *exp num 1*, le résultat vaut 0 si le bit correspondant du mot évalué par *exp num 2* est à zéro ou si le bit testé vaut 0 quand le bit correspondant de l'octet de valeur *exp num 3* vaut 0 ou si le bit testé vaut 1 quand le bit correspondant de l'octet de valeur *exp num 3* vaut 0.
- pour chacun des bits de l'octet adressé par *exp num 1*, le résultat vaut 1 si le bit correspondant de l'octet de valeur *exp num 2* vaut 1 et si le bit testé vaut 1 quand le bit correspondant de l'octet de valeur *exp num 3* vaut 1 et vaut 0 quand le bit correspondant de l'octet de valeur *exp num 3* vaut 0.

Notons *t₀ t₁ ..., t₇* les valeurs des bits de l'octet testé adressé par *exp num 1*. Notons *m₀ m₁ ..., m₇* les valeurs des bits de l'octet de valeur *exp num 2*. Notons *a₀ a₁ ..., a₇* les valeurs des bits de l'octet de valeur *exp num 3*. Ces dernières valeurs valent 0 si *exp num 3* est absente.

Pour chacune des valeurs de l'indice *i* comprises entre 0 et 7, on a :

m_i = 0 alors *r_i* = 0, *r_i* notant le résultat de l'opération logique.
m_i ≠ 0 alors si *t_i* ≠ *a_i* *r_i* = 1
si *t_i* = *a_i* *r_i* = 0

Si l'un au moins des *r_i* est non nul, alors la condition de fin d'attente est réalisée et le programme continue en séquence.

- il faut noter qu'une expression du type WAIT X, 0 fait attendre le programme indéfiniment puisque tous les bits du masque de test sont à 0. Pour sortir de cette attente, il faut faire *ctrl*/C et l'utilisateur reprend le contrôle de la machine.

Exemples :

WAIT X, 255, 0

signifie attendre jusqu'à ce qu'un au moins des bits de l'octet d'adresse décimale X soit à 1.

WAIT X, 255

signifie la même chose que ci-dessus : le troisième paramètre vaut 0 par défaut.

WAIT X, 255, 255

signifie attendre jusqu'à ce qu'un au moins des bits de l'octet d'adresse décimale X soit à 0.

WAIT X, 1, 1

signifie attendre jusqu'à ce que le bit 0 (bit de poids le plus faible) soit à 0, quel que soit l'état des autres bits de l'octet d'adresse décimale X.

WAIT X, 3, 2

signifie attendre jusqu'à ce que soit le bit 0 soit à 1, soit le bit 1 soit à 0, soit les deux conditions ci-dessus soient réalisées, les bits testés appartenant à l'octet d'adresse décimale X.

Le programme suivant attend qu'un caractère soit frappé au clavier et attend ensuite qu'un caractère compris entre le caractère @ et 0 soit frappé au clavier, c'est à dire qu'un code décimal compris entre 64 et 80 soit entré.

10 POKE - 16 368, 0	<i>remet à 0 l'indicateur de</i>
20 WAIT - 16 384, 128	<i>caractères ; attend la frappe</i>
30 WAIT - 16 384, 192, 128	<i>d'un caractère ; attend la</i>
40 PRINT " ? "	<i>frappe d'un caractère dont</i>
50 GOTO 10	<i>le bit 6 est à 1 et le bit 7 à 0.</i>

Note :

Pour calculer la valeur décimale de l'octet décrivant une configuration binaire, il suffit d'appliquer la règle de calcul suivante. La suite de bits étant décrite par *b_k*, variant de 0 à 7, 0 indiquant le bit le plus à droite ou le bit de poids le plus faible, l'octet se représente en binaire de la façon suivante (les *b_i* valant 0 ou 1) :

b₇ b₆ b₅ b₄ b₃ b₂ b₁ b₀

La valeur décimale est alors :

b₇ × 128 + *b₆* × 64 + *b₅* × 32 + *b₄* × 16 + *b₃* × 8 + *b₂* × 4 + *b₁* × 2 + *b₀*

7. LES INSTRUCTIONS D'ENTREES/SORTIES.

Ce groupe d'instructions permet à un programme d'échanger des données non plus à l'intérieur de la mémoire mais avec "l'extérieur". Cet "extérieur" dépend de la configuration dans laquelle est utilisée la machine et les dispositifs de communication de la machine avec cet extérieur sont appelés périphériques. Physiquement ces périphériques sont connectés à la machine par l'intermédiaire d'interfaces qui sont enfichés sur des prises numérotées : à chaque prise correspondant un numéro qui identifie le *canal* par lequel les données transitent.

Les échanges avec l'extérieur sont bidirectionnels, c'est à dire que la machine peut à la fois envoyer des données vers l'extérieur ou bien en recevoir. Néanmoins un certain nombre de dispositifs externes ne sont conçus que pour l'un des deux modes de fonctionnement : un traceur de courbe est a priori un dispositif qui ne peut que recevoir des données, un clavier un dispositif qui ne peut qu'en émettre, un lecteur de cassette peut effectuer les deux opérations.

Les instructions de gestion des entrées/sorties sont donc prévues pour permettre le dialogue avec ces dispositifs externes.

Parmi tous ces organes d'entrées/sorties, il en est un que la machine favorise car c'est le périphérique avec lequel on dialogue par défaut : il s'agit de l'ensemble clavier pour l'introduction des données, écran pour la visualisation des résultats.

7.1 INPUT

Objet : permettre d'assigner des valeurs à des variables à partir de l'organe d'entrée courant.

Syntaxe : INPUT (" *texte* "); *variable 1*, (*variable 2*, (*variable 3*, (...)))

Commentaires : ● l'instruction INPUT demande que soient présentées à l'organe d'entrée défini avant l'opération des valeurs compatibles avec le type de données de la liste de variables et en nombre égal, séparées par des virgules ou des retours chariot (provoqués par la frappe de la touche RETURN).

● le texte facultatif noté *texte* permet d'afficher un commentaire pour identifier les données demandées. Si il est présent, il doit être situé entre parenthèses, ne contenir ni cotes " ni retour chariot RETURN, et être suivi d'un point virgule ;. Ce commentaire ne peut que suivre le mot INPUT et ne peut pas être répété pour chaque variable demandée. Si ce commentaire est absent, l'interpréteur BASIC APPLESOFT imprime en début de ligne un point d'interrogation pour signaler que le programme demande des données.

● si *variable 1*, *variable 2*, ..., sont des variables numériques, l'ordre INPUT acceptera uniquement en entrée des valeurs réelles ou entières. Les caractères espace (ou blanc), +, -, E, . sont des caractères valides pour des données numériques. Si la donnée est limitée à l'un de ces caractères, la valeur réelle assignée est zéro. Les blancs sont ignorés dans l'énoncé d'un nombre, quelle que soit leur position.

● si l'une des variables *variable 1*, *variable 2*, ... est une variable alphanumérique, l'ordre INPUT acceptera pour la donnée correspondante :

- soit une chaîne de caractères situés entre cotes " : cette chaîne peut contenir des caractères quelconques à l'exception d'une cote " (qui doit être suivie uniquement de blancs pour être valide), de *ctrl M* (ce dernier caractère est identique au caractère obtenu par la frappe de la touche RETURN, donc ne peut être situé dans une chaîne de caractère entrée au clavier puisque par essence il la termine). Si le séparateur entre les différentes entrées est obtenu par la touche RETURN, on peut omettre la cote finale.

- soit une chaîne de caractères telle quelle, sans délimiteurs. Cette chaîne peut alors contenir des cotes dans son énoncé sauf en première position car l'interpréteur croirait avoir affaire à une chaîne de caractères situés entre cotes (voir ci-dessus). Les blancs situés avant le premier caractère sont ignorés, par contre les blancs situés entre le dernier caractère et le séparateur de données (virgule ou touche RETURN) sont conservés. Les seuls caractères interdits sont la virgule (qui est interprétée comme un séparateur), les deux points : qui délimitent un champ de commentaires pour les données et à partir desquels tous les caractères sont ignorés y compris les virgules. Après les deux points dans une chaîne de caractères non délimitée par des cotes, seul un retour chariot RETURN peut être utilisé comme séparateur. Il apparaît

un message indiquant qu'une partie de la donnée n'a pas été prise en compte : ?EXTRA IGNORED et l'exécution du programme continue. Comme ce message n'est pas un message d'erreur, il n'est pas affecté par l'instruction ONERR GOTO Les caractères *ctrl M* ne peuvent pas être utilisés dans un ordre INPUT : *ctrl X* ne peut pas être utilisé en entrée au clavier car il indique que tous les caractères entrés auparavant ne sont pas valides et que l'on recommence l'introduction de la donnée ou des données à partir du dernier retour chariot RETURN ; *ctrl M* est identique au retour chariot RETURN et donc termine nécessairement l'introduction de la donnée en cours.

● si on a introduit moins de données qu'il n'en est demandé dans l'instruction contenant l'ordre INPUT en cours, l'interpréteur BASIC APPLESOFT demande le complément par l'affichage de ??.

● si on introduit sur une même ligne plus de données qu'il n'est prévu d'en fournir pour l'instruction INPUT en cours (ces données sont alors séparées par des virgules), l'interpréteur BASIC APPLESOFT se limitera à prendre en compte le nombre exact de données dont il a besoin et après avoir affiché le message ?EXTRA IGNORED, il continuera normalement l'exécution du programme.

● le type des diverses variables dans un même ordre INPUT peut être quelconque, c'est à dire qu'on peut mélanger dans la liste des variables de l'ordre INPUT des variables entières, réelles et alphanumériques.

● le type des données fournies en entrée doit être compatible avec le type des variables dans le même ordre dans la liste située dans l'instruction INPUT. Si l'option ONERR GOTO ... n'est pas utilisée, une erreur dans le type de donnée cause l'apparition du message ?REENTER et l'exécution de l'instruction INPUT de nouveau. La présence de caractères autres que des blancs entre la cote finale d'une chaîne de caractères et le séparateur de données cause les mêmes effets.

● si l'instruction ONERR GOTO ... a été utilisée pour traiter les erreurs, on notera que si l'instruction de retour vers l'instruction INPUT en erreur est un GOTO la 87^{ème} erreur causera un abandon du programme BASIC et un débranchement vers le Moniteur. Si l'instruction de retour vers l'ordre INPUT est une instruction RESUME, aucun fonctionnement anormal n'est à signaler. On se reportera avec intérêt à la description de l'instruction ONERR GOTO au 6.8 qui indique un moyen d'éviter ces inconvénients.

● la frappe de la touche RETURN en réponse à une demande de donnée a pour effet de causer l'apparition du message ?REENTER si la variable concernée est numérique et de donner à une variable alphanumérique la valeur de la variable alphanumérique de longueur nulle.

● les caractères virgule, deux points introduits comme seule réponse à une demande de donnée donnent à une variable numérique la valeur nulle et à une variable alphanumérique la valeur de la variable alphanumérique de longueur nulle.

● on remarquera que la chaîne de caractères format le commentaire facultatif doit être suivie d'un point virgule si elle est présente tandis que les différentes variables doivent être séparées par des virgules.

● l'instruction INPUT ne peut être utilisée que dans un programme exécuté en mode différé. Son utilisation en mode immédiat cause l'apparition du message ?ILLEGAL DIRECT ERROR. A noter que l'instruction de l'instruction INPUT en mode direct n'est pas d'une grande utilité !

● *ctrl C* peut interrompre un ordre INPUT si et seulement si il est le premier caractère introduit en réponse à l'instruction INPUT et s'il est suivi d'un retour chariot RETURN. Le programme s'interrompt alors, mais ne peut être relancé par l'instruction CONT. Toute tentative de le faire repartir engendre le message ?SYNTAX ERROR. Si *ctrl C* n'est le premier caractère introduit, il est considéré comme un caractère quelconque.

7.2 GET

Objet : aller chercher un caractère sur l'organe d'entrée courant sans l'afficher et le prendre en compte sans indicateur de fin de message, c'est à dire sans avoir à frapper la touche RETURN.

Syntaxe : GET *VARIABLE*

Commentaires : ● GET ne peut être exécuté qu'en mode d'exécution différé.

● si *variable* est une variable alphanumérique, le caractère obtenu est le dernier caractère frappé au clavier mais avec les exceptions suivantes :

- *ctrl a* retourne le caractère de longueur nulle

- la flèche orientée vers la gauche peut retourner elle aussi le caractère de longueur nulle si aucun caractère autre que la flèche gauche n'a été frappé auparavant. *ctrl H* qui est identique à la flèche gauche obtient les mêmes résultats.

● *ctrl C* est considéré comme un caractère quelconque et ne peut en aucun cas interrompre un programme exécutant une instruction GET.

● L'interpréteur BASIC APPLESOFT n'a pas été conçu pour exécuter une instruction GET portant sur des variables numériques. Il est néanmoins possible d'utiliser l'instruction GET quand *variable* est une variable numérique, entière ou réelle. Les contraintes d'emploi sont cependant sévères :

- si le caractère frappé est une virgule ou deux points, on reçoit le message ?EXTRA IGNORED (qui n'est pas compréhensible sauf si l'on se râorte à la description de l'instruction INPUT) et le chiffre retourné est zéro.

- les caractères +, -, E, ., blanc ou espace et *ctrl a* retournent la valeur zéro.

- tout caractère non numérique différent de ceux cités ci-dessus provoque l'affichage du message ?SYNTAX ERROR et subséquemment, l'arrêt du programme. Si l'instruction ONERR GOTO a été utilisée pour traiter les erreurs, conjointement avec l'instruction RESUME pour reprendre le cours normal du programme, deux erreurs consécutives sur une instruction GET font que le système se suspend indéfiniment : seule la séquence RESET *ctrl C* peut rendre le contrôle de la machine à l'utilisateur. Si une instruction GOTO est utilisée à la place de RESUME, le système passe sous le contrôle du moniteur en abandonnant l'exécution du programme BASIC en cours après la 43^{ième} erreur sur une instruction GET (les erreurs n'étant pas nécessairement consécutives). Ces incidents peuvent être évités en suivant la méthode indiquée au 6.8.

● ces limitations d'emploi de l'instruction GET avec des variables numériques font qu'il est souvent plus simple et plus sûr d'utiliser GET avec une variable alphanumérique dont la valeur numérique peut être testée et si elle est correcte convertie en nombre par la fonction VAL.

Exemple :

```
10 GET A$           prendre un caractère au
                    clavier et le mettre dans
20 A = VAL (A$)     la variable AS ; convertir
                    cette dernière en un nombre
30 IF A <> 0 THEN PRINT A si possible ; si le nombre
                    est différent de 0, l'imprimer ; aller en 10
40 GOTO 10
```

7.3

PRINT

Objet : permettre d'imprimer ou d'afficher sur l'organe courant de sortie la valeur d'une liste de variables ou de constantes.

Syntaxe : PRINT (*exp 1* (; *exp 2* (; ...))) (;)

Commentaires : ● lors de l'écriture d'un programme, PRINT peut être aussi codé ?. Il est néanmoins converti en PRINT lors de l'impression du programme par l'instruction LIST.

● les expressions *exp 1*, *exp 2*, ... sont quelconques et l'instruction PRINT affiche leur valeur. Si ces expressions sont composées des fonctions d'espacement SPC et de tabulation TAB, on se reportera plus loin pour une description précise de ces fonctions qui ne peuvent être utilisées qu'à l'intérieur d'un ordre PRINT. La position d'affichage des expressions dépend des longueurs des affichages des variables et expressions de la liste précédemment imprimées et des opérateurs de tabulation et espacement, les séparateurs virgule et point-virgule faisant partie de ceux-ci. La disposition des résultats est discutée plus loin.

● si *exp 1* est une expression prenant des valeurs alphanumériques, la chaîne de caractères qu'elle constitue est affichée telle quelle. On prendra garde à la présence éventuelle de caractères de contrôle qui peuvent des effets non désirés : si par exemple la chaîne de caractères contient plus de 24 *ctrl J* (qui produisent un saut de ligne), les premiers caractères inscrits seront perdus !

● si *exp 1* est une expression prenant des valeurs numériques, donc si elle est une expression numérique ou logique, le format d'impression dépend de la valeur absolue du nombre à afficher.

- si la valeur absolue du nombre est comprise entre 0 et 0.01, ce dernier nombre exclu, les nombres sont exprimés dans la notation en virgule flottante, c'est à dire sous la forme d'une mantisse de 9 chiffres et d'un exposant entier ; par exemple .12345 E - 12 qui représente le nombre 1,2345 10⁻¹². De même les nombres d'une valeur absolue supérieure à 999 999 999 .2 sont exprimés en virgule flottante ; ainsi, un nombre d'un milliard sera écrit 1 E 9. Si la valeur absolue est comprise entre 0.01 et 0.0 999 999 999, les nombres réels sont imprimés en virgule fixée, donc sous la forme courante avec 10 chiffres au total et au plus, les zéros après la virgule étant supprimés sauf le premier d'entre eux ; par exemple .020.

Si la valeur absolue du nombre est comprise entre 0.1 et 999 999 999.2, le nombre est imprimé en virgule fixe avec 9 chiffres au plus, les zéros à gauche du point décimal étant supprimés. Si un nombre ne possède que des zéros à droite du point décimal, ce dernier est supprimé et le nombre est imprimé comme un nombre entier.

- si le nombre est négatif, il est précédé du signe -.

● si les éléments de la liste d'expressions suivant le mot PRINT ne sont pas séparés par des virgules ou des points-virgules, les valeurs des expressions sont concaténées si aucun problème de syntaxe n'apparaît. Dans le cas contraire, on obtient le message ?SYNTAX ERROR. Si un point ne peut pas être interprété comme un point décimal, il est évalué à zéro. Les exemples plus bas illustrent la faculté de concaténation de l'instruction PRINT.

● si deux éléments de la liste à imprimer sont séparés par un point-virgule, ces deux éléments sont concaténés.

● si un point-virgule est le dernier élément d'une instruction PRINT, le curseur reste à la dernière position affectée, sans changement à la fin de l'ordre PRINT. Ceci permet de concaténer des éléments même s'ils ne sont pas imprimés ou affichés par la même instruction.

● si deux éléments de la liste sont séparés par une virgule, après l'impression du premier de ces deux éléments, le curseur va se positionner au début du prochain champ de tabulation : ces champs sont définis de la façon suivante ; le premier champ occupe les 16 premières positions de la fenêtre d'impression, le second champ les 16 suivantes et ainsi de suite. Le deuxième champ de tabulation est utilisable si rien n'est imprimé ou affiché en 16[°] position, le troisième n'est utilisable si rien n'est affiché ou imprimé en 32[°] position et si la fenêtre d'impression est d'au moins 33 caractères de long, et ainsi de suite en allant de 16 en 16.

● si le dernier caractère de l'instruction PRINT est une virgule, le curseur se positionnera au début du champ de tabulation suivant et sa position restera inchangée à la fin de l'ordre PRINT.

● si le dernier caractère de l'instruction PRINT est différent d'une virgule ou d'un point-virgule, la fin de traitement de l'instruction PRINT se traduit par un saut de ligne et par un retour du curseur au début de la ligne suivante.

● si l'une des expressions de la liste est composée de la fonction de tabulation TAB, le curseur se déplace à la position indiquée par l'évaluation du paramètre de la fonction TAB si cette position est supérieure à la position courante du curseur, sinon TAB reste sans effet. Il faut pour faire "reculer" le curseur utiliser l'instruction HTAB et ceci à l'extérieur de l'instruction PRINT. La syntaxe de la fonction TAB est

TAB (*exp num*)

où *exp num* doit être comprise entre 0 et 255 pour éviter l'affichage du message ?ILLEGAL QUANTITY ERROR. TAB (0) PLACE LE CURSEUR A LA 256[°] position.

● lorsque le curseur atteint l'extrémité droite de la fenêtre définie pour l'écran, il saute à la ligne suivante et se positionne au début de cette ligne. Ainsi dans les conditions normales d'emploi où la largeur de l'écran est définie à 40 caractères, l'instruction PRINT TAB (41) placera le curseur au début de la ligne suivante.

● si l'une des expressions de la liste suivant l'ordre PRINT est composée de la fonction d'espacement SPC, le curseur se déplace d'un nombre de positions égal à l'évaluation du paramètre de la fonction SPC. La

syntaxe de la fonction d'espacement SPC est la suivante :

SPC (*exp num*)

où *exp num* doit prendre des valeurs comprises entre 0 et 255 pour éviter l'affichage du message ?ILLEGAL QUANTITY ERROR. SPC (0) reste sans effet.

● on notera la différence entre les fonctions TAB et SPC. Tandis que TAB (X) place le curseur à la position X sur la ligne à partir du début de celle-ci, SPC (X) laisse un espace de X blancs entre la dernière position du curseur et la nouvelle position. TAB effectue un déplacement absolu du curseur alors que SPC effectue un déplacement relatif.

● on rappelle ici les instructions qui permettent de modifier la taille et la position de la fenêtre d'impression :

- POKE 32, *position marge gauche* *valeur par défaut 0*
- POKE 33, *nombre de caractères par ligne* *valeur par défaut 40*
- POKE 34, *position première ligne* *valeur par défaut 0*
- POKE 35, *nombre de lignes par page* *valeur par défaut 24*

● si le nombre de caractères par ligne fixé par une instruction POKE est inférieur à 33, des caractères peuvent être imprimés à l'extérieur de la fenêtre par l'utilisation du troisième champ de tabulation, c'est à dire lors de l'impression de la troisième expression de la liste, si ces expressions sont séparées par des virgules.

● le nombre d'expressions, de séparateurs et de fonctions de tabulation et d'espacement utilisés dans une instruction PRINT est quelconque et n'est limité que par la longueur physique de la ligne d'instruction soit 239 caractères.

Exemples :

]10 PRINT 1, " TEST " ; 2

]20 A = 1 : B = 2 : C = 3 :

C (4) = 5 : C5 = 7

]30 PRINT 1/3 (2×4) 51

]40 ?1 (A) 2 (B) 3 C (4) C5

]50 PRINT 3.4.5.6.,

]60 PRINT A. " B. " C.4

]RUN

exécution du programme

1 TEST 2

la deuxième constante est imprimée à partir de la position 16. Les valeurs numériques sont concaténées.

.333333333851

1122357

PRINT reconnaît les parenthèses délimitant des indices.

3.4.5.60 10B.3.4.

Les points non décimaux sont pris pour des zéros. L'instruction PRINT de la ligne 60 se place au début du 2° champ de tabulation, le dernier caractère de la précédente instruction PRINT étant une virgule.

7.4

DATA

Objet : permettre de stocker des données qui vont être utilisées dans le programme et dont les valeurs sont assignées aux variables par des instructions READ.

Syntaxe : *nombre entier*
DATA *nombre réel* (, ...)
 chaîne de caractères

Commentaires : ● l'ordre DATA sert simplement à indiquer que les éléments qui suivent sont des données et, en conséquence peut être placé à un endroit quelconque de la liste d'instructions. Néanmoins, si plusieurs instructions DATA sont utilisées dans le programme, les données seront lues dans l'ordre des instructions DATA.

● les instructions DATA créent une liste de données de type quelconque dont les valeurs sont assignées à des variables du type correspondant par l'instruction READ. Les valeurs qui constituent la liste de données d'un ordre DATA sont de trois types :

- valeurs entières
- valeurs réelles
- chaînes de caractères dont le format est double :

- la chaîne de caractères est délimitée par des cotes " et dont ne peut pas contenir de cotes en son sein. Tous les caractères sont autorisés sauf *ctrl* X (qui annulent ce qu'on vient de rentrer au clavier) ni *ctrl* M (qui est identique à RETURN et donc termine la ligne d'instruction). Si une cote terminant une chaîne de caractères est le dernier caractère d'une ligne d'instruction, elle peut être omise.

- la chaîne de caractères n'est pas délimitée par des cotes. Elle peut contenir des cotes en son sein sauf en première position, mais la virgule et les deux points sont interdits. La virgule est considérée comme un séparateur de données et les deux points délimitent un champ de commentaires pour les données : les caractères, y compris les virgules, sont ignorés après les deux points ; il faut dans utiliser un autre ordre DATA pour inscrire la suite des données. Les caractères *ctrl* X et *ctrl* M sont interdits pour les mêmes raisons que plus haut.

● les séparateurs de données à l'intérieur d'une instruction DATA sont limités à la virgule.

● les données suivent les mêmes règles que les réponses à des ordres INPUT, à l'exception des deux règles suivantes :

- les deux points sont interdits dans des données numériques ;
- *ctrl* C est un caractère comme les autres et ne peut, même en première position interrompre le programme.

● les données dans un ordre DATA sont d'un type quelconque et peuvent être mélangées sans problèmes.

● si la liste d'éléments d'un ordre DATA contient un élément " non-existant ", alors la valeur est assignée à la variable correspondante si cette dernière est numérique et la chaîne de caractères de longueur nulle est assignée à la variable correspondante si cette dernière est alphanumérique. Un élément " non existant " est défini par l'occurrence des circonstances suivantes :

- il n'y a pas de caractères non blancs entre le mot DATA et le retour du chariot provoqué par la touche

RETURN ;

- une virgule est le premier caractère non blanc suivant le mot DATA ;
- il n'y a pas de caractères non blancs entre deux virgules consécutives ;

- une virgule est le dernier caractère non blanc précédant la fin de la ligne d'instruction matérialisée par la frappe de la touche RETURN.

● DATA ne s'utilise qu'en mode d'exécution différé. Néanmoins son utilisation en mode immédiat ne cause pas d'erreur mais reste sans effet.

● la seule erreur causée par un ordre DATA est la suivante : si une cote apparait au milieu d'une chaîne de caractères délimitée par des cotes, il y apparition du message ?SYNTAX ERROR.

Exemples :

On se rapportera aux exemples de l'instruction READ qui est indispensable à l'utilisation des données stockées par un ordre DATA.

7.5 READ

Objet : assigner à des variables prises dans une liste les valeurs correspondantes prises dans une liste de données stockées par une instruction READ.

Syntaxe : READ variable 1 (, variable 2 (, ...))

Commentaires : ● si variable 1 est la n^{ième} variable rencontrée dans un ordre READ depuis le début du programme, elle recevra la valeur de la n^{ième} données stockée dans un ordre DATA (ou ordres DATA successifs) si le pointeur de la liste de données n'a pas été modifié par un ordre RESTORE. En effet, au début du programme, ce pointeur indique que la première donnée disponible pour lecture par une instruction READ est la première donnée de la première instruction DATA. Après l'exécution de cet ordre READ, le pointeur a avancé du nombre de position égal au nombre de variables de la liste suivant READ, et la première variable de la liste a reçu la valeur de la première donnée, la seconde variable, la valeur de la seconde donnée, et ainsi de suite jusqu'à épuisement de la liste de variables dans l'ordre READ ou épuisement des données dans l'ensemble des instructions DATA.

● si la n^{ième} variable est d'un type incompatible avec celui de la n^{ième} donnée, il y a affichage du message ?SYNTAX ERROR IN numéro de la ligne de l'ordre DATA contenant la donnée erronée.

● si le nombre de données demandées par l'ensemble des ordres READ est supérieur au nombre de données stockées par les instructions DATA, c'est à dire lorsque le pointeur a dépassé le dernier élément disponible dans les instructions DATA, il y a affichage du message ?OUT OF DATA ERROR IN numéro de la ligne de l'ordre READ demandant la première donnée inexistante. (le nombre de demandes de données peut cependant être supérieur au nombre de données si l'on repositionne le pointeur au début de la liste pour relire les données par un ordre RESTORE).

● en mode d'exécution immédiat, on ne peut lire les données que dans des ordres DATA d'un programme en mémoire, mais le pointeur de la liste de données est là où une exécution précédente l'a laissé. Si aucun ordre DATA n'est présent dans le programme en mémoire ou si aucun programme n'est couramment en mémoire ou si la liste des données a été épuisée par le programme en mémoire, il y a affichage du message ?OUT OF DATA ERROR.

● la commande RUN positionne le pointeur de liste de données sur le premier élément d'une instruction DATA (s'il en existe une).

● les variables constituant la liste sont séparées par des virgules.

Exemples :

10 READ A, A\$	charger A avec la première donnée, A\$ avec la seconde
20 DATA 10, DIX,	
30 READ C	charger C avec la troisième donnée
40 DATA " FIN "	
50 PRINT A : PRINT A\$: PRINT C	imprimer A, A\$, C il reste une donnée inutilisée.
RUN	Exécution du programme

10	valeur de A
DIX	valeur de A\$
0	valeur de C
*READ A\$: PRINT A\$	exécution du programme en mode immédiat : charger AS avec la donnée sur laquelle on s'est arrêté dans le programme précédent et l'imprimer.
FIN	

7.6 RESTORE

Objet : repositionner le pointeur de liste de données sur le premier ordre DATA du programme.

Syntaxe : RESTORE

Commentaires : ● RESTORE n'a ni paramètres ni options. Il déplace le pointeur de liste de données au début de la première instruction DATA.

elle peut être employée en mode immédiat pour positionner le pointeur laissé à la position courante par un programme.

Exemple :

10 DATA 1, 2, 3	le pointeur pointe sur 1.
20 READ A, B	le pointeur pointe après cette instruction sur 3
30 RESTORE	le pointeur pointe de nouveau sur 1
40 PRINT A, B	
50 READ C	le pointeur pointe après le READ sur 2
RUN	Exécution du programme.
1 2	Valeurs de A et B
*READ A : RESTORE : ?A, : READ A : ?A	Programme en mode immédiat. Le pointeur pointe d'abord sur 2 là où l'a laissé le programme précédent puis RESTORE le ramène sur le premier élément de la liste DATA du programme couramment en mémoire.
2 1	

7.7 STORE

Objet : stocker sur cassette le contenu d'un tableau de valeurs numériques.

Syntaxe : STORE nom de tableau numérique

* Avec le système disque DOS, cette instruction provoque l'apparition du message d'erreur, NOT DIRECT COMMAND

Commentaires : ● *nom de tableau numérique* étant le nom d'un tableau de dimensions $n1, n2, n_p$ l'instruction STORE effectue les opérations suivantes :

- envoyer un signal sonore signalant le début de l'enregistrement ;
- envoyer vers le magnétophone à cassette les $(n1 + 1) \times (n2 + 1) \times \dots \times (n_p + 1)$ éléments du tableau en faisant d'abord varier le dernier indice, puis l'avant dernier, ..., c'est à dire que le premier indice est le plus intérieur (voir en 2.5.5) ; par exemple si A est un tableau dimensionné à 1, 1 et tel que : A (0,0) = 0 ; A (1,0) = 1 ; A (0,1) = 2 ; A (1,1) = 3, les nombres qui seront stockés sur la cassette seront les nombres *réels* 0,1,2,3 dans cet ordre.

- envoyer un signal sonore de fin d'enregistrement.
 - si le tableau est un tableau de nombres entiers, ces derniers sont convertis en nombre réels.
 - le nom du tableau ni ses dimensions n'étant stockés sur la cassette, il est possible d'utiliser ces données sous un autre nom de tableau après chargement par l'instruction RECALL. On se reportera à cette instruction pour toute information concernant l'exploitation des données stockées.

- si *nom de tableau numérique* n'a été utilisé nulle part auparavant dans le programme, soit dans un ordre DIM, soit dans une assignation de valeur à une variable indicée, il y a affichage du message : ?OUT OF DATA ERROR

- si une variable simple possède le même nom que le tableau à stocker sur cassette, l'interpréteur BASIC APPLESOFT fait la différence entre les deux types de variables numériques et aucune confusion n'est à craindre (voir exemple).

- STORE peut être utilisé en mode immédiat, mais le nom du tableau doit être un nom de tableau dimensionné dans un programme exécuté précédemment ou dans le programme courant.

- les variables alphanumériques, même indicées ne peuvent être directement stockées sur cassette par l'instruction STORE. Il faut auparavant convertir les caractères des chaînes de caractères en valeurs numériques placées dans un tableau numérique (de valeurs entières pour limiter l'occupation de la mémoire). On peut ensuite évidemment stocker ce tableau sur cassette par l'intermédiaire de l'instruction STORE. La fonction ASC permet de convertir les caractères alphanumériques en leurs codes numériques internes.

- on prendra garde à ne pas utiliser de variables dont le nom commence par STORE car l'interpréteur BASIC APPLESOFT tentera de stocker sur cassette le tableau dont le nom commence par les DEUX caractères suivant le mot STORE. S'il existe un tel tableau, l'interpréteur le stockera sur cassette (ou du moins enverra vers le magnétophone les signaux correspondant) avant de détecter une éventuelle erreur de syntaxe. Si un tel tableau n'existe pas, il emmettra le message ?OUT OF DATA ERROR et donc interrompra là l'exécution du programme.

Exemples :

0 STORE XX = 0

10 ...

RUN *exécution du programme*

?OUT OF DATA ERROR IN 0 *erreur détectée : le tableau n'existe pas nouveau programme*

0 DIM XX (10)

10 STORE XX = 0

20 ...

RUN *exécution du programme*

?SYNTAX ERROR IN 10 *après avoir "stocké" le tableau XX sur cassette, une erreur est détectée : = 0 n'est pas une instruction valide !*

- si l'on dispose d'un système équipé des disquettes, l'utilisation de cette instruction présente peu d'intérêt car les données ainsi stockées ne sont pas identifiées, et leur exploitation par l'instruction RECALL nécessite la connaissance de la taille du tableau stocké. D'autre part, la difficulté de stocker des données alphanumériques rend peu commode cette instruction. Un système disque permet au contraire de stocker des données quelconque sur disquettes en leur donnant un nom, ce qui résoud les problèmes d'identification.

Exemples :

10 A (1) = 10

20 A = 12

30 STORE A

40

Les nombres stockés sur la cassette sont les suivants :
0 10 0 0 0 0 0 0 0 0 0

En effet, le tableau est dimensionné à 10 par défaut, donc contient 11 éléments et seul l'élément d'indice 1, donc le second est non nul.

La variable réelle non indicée A n'intervient dans la liste des nombres stockés, l'interpréteur BASIC APPLESOFT différenciant les deux types de variables.

7.8

RECALL

Objet : lire sur cassette les valeurs numériques *réelles* d'éléments d'un tableau précédemment chargées sur cassette par une instruction STORE.

Syntaxe : RECALL *nom de tableau numérique*

Commentaires : ● *nom de tableau numérique* étant le nom d'un tableau à valeurs réelles ou entières, ayant été précédemment dimensionné, soit par une instruction DIM, soit par une assignation de valeur à l'un de ses éléments, aux dimensions $n1, n2, n_p$ étant le nombre de dimensions du tableau, l'instruction RECALL charge au fur et à mesure les éléments de tableau à partir des valeurs lues séquentiellement sur la bande magnétique. L'ordre de chargement des valeurs du tableau est celui du stockage par l'instruction STORE. Ainsi, si la taille* du tableau chargé par RECALL est identique à celle du tableau stocké par STORE, et si les dimensions sont identiques, tous les éléments du tableau initial se retrouvent dans le tableau final et à une place correcte. Si la taille du tableau initial est supérieure à celle du tableau récepteur, il y a affichage du message ?OUT OF MEMORY ERROR, ce qui signifie que l'on a épuisé l'espace mémoire disponible pour le tableau sans pouvoir charger toutes les valeurs stockées sur la bande magnétique. Si la taille du tableau récepteur est supérieure à celle du tableau initial, un commentaire d'avertissement apparaît : ERR, mais l'exécution du programme continue normalement, et les éléments supplémentaires sont laissés inchangés.

Si les dimensions ne sont pas identiques, bien que les tailles soient égales ou suffisantes pour recevoir tous les éléments, on risque de retrouver les éléments du tableau initial dans un *désordre* certain. En effet, si $m1, m2, \dots, m_q$ étant le nombre de dimensions, sont les dimensions du tableau stocké par STORE, l'élément de ce tableau d'indices $i1, i2, \dots, i_q$ est enregistré sur la bande magnétique à la $i1 + i2 \times m1 + \dots + i_q \times m1 \times m2 \times \dots \times m_{q-1}$ ième position. Si $m1 = n1, m2 = n2, \dots, m_{q-1} = n_{q-1}$ alors la valeur de l'élément du tableau initial d'indices $i1, i2, \dots, i_q$ sera affectée à l'élément de **mêmes indices** ; si p est supérieur à q ou si seulement n_q est supérieur à m_q alors les éléments résiduels sont inchangés. On voit par contre le problème qui se pose si les deux listes de dimensions diffèrent par un élément qui ne soit pas au moins le q ième : les calculs sont faisables, mais assez vite pénibles !

*taille = produit des dimensions

● en résumé :

- si les tailles sont identiques et les dimensions identiques, les éléments du tableau initial se retrouvent dans le tableau final à la même place (donc identifiés par les mêmes indices).
- si le tableau récepteur est de taille supérieure au tableau initial, mais si les $p - 1$ premières dimensions sont identiques (p étant le nombre de dimensions du tableau initial), il y a affichage de l'avertissement ERR mais les éléments de mêmes indices ont même valeur et les éléments du tableau récepteur n'ayant pas d'homologues dans le tableau initial sont inchangés.
- si le tableau récepteur est de taille supérieure à celle du tableau initial, ou si les tailles sont égales mais si les dimensions sont différentes, il y a affichage du message ERR et les valeurs du tableau initial se retrouvent dispersées dans l'ensemble du tableau récepteur.
- si la taille du tableau récepteur est inférieur à celle du tableau initial, il y a affichage du message ?OUT OF MEMORY ERROR et arrêt du programme.

● l'instruction RECALL émet un signal sonore pour indiquer le début de la lecture des données enregistrées : si le lecteur de cassette n'est pas en fonctionnement, l'interpréteur BASIC APPLESOFT attend qu'on lui fournisse les données. Si le lecteur n'est pas prêt et si l'on désire interrompre l'instruction RECALL, il faut utiliser la séquence RESET *ctrl* C pour reprendre le contrôle de la machine. A la fin de l'instruction, un signal sonore est émis pour indiquer la fin de l'opération.

● RECALL peut être utilisé en mode immédiat à condition que le tableau récepteur existe dans un programme exécuté précédemment ou dans le programme courant. Sinon, il y a affichage du message ?OUT OF DATA ERROR.

● on prendra garde à ne pas utiliser de variables commençant par RECALL sous peine de voir apparaître le message ?OUT OF DATA ERROR si les deux caractères suivant RECALL ne sont pas les premiers caractères d'un tableau existant, et de voir dans le cas contraire le système attendre sereinement des données sur cassette qui ne viendront jamais ! Il faut dans ce cas utiliser la séquence d'interruption RESET *ctrl* C pour reprendre le contrôle de la machine.

Exemples :

exemple 1 :	
10 DIM A (5,5,5)	<i>dans ce programme, on voit</i>
20 ...	<i>l'utilisation normale de l'</i>
30 STORE A	<i>instruction RECALL</i>
40 DIM B (5,5,5)	<i>associée à une instruction STORE</i>
50 RECALL B	
exemple 2 :	
10 DIM A (1,1)	<i>Ce programme montre comment</i>
20 A (0,0) = 0 : A (1,0) = 1	<i>bien que les tableaux aient</i>
30 A (0,1) = 2 : A (1,1) = 2	<i>dimensions différentes</i>

40 STORE A	<i>les éléments peuvent se</i>
50 DIM B (3)	<i>correspondre (dans un cas simple)</i>
60 RECALL B	
70 PRINT B (0), B (1), B (2), B (3)	
RUN	<i>exécution du programme ci-dessus</i>

0 1 2 3

7.9 IN #

Objet : changer le numéro du canal par lequel sont fournies les données à la machine, et en conséquence l'organe d'entrée est changé.

Syntaxe : IN # *numéro de canal*

Commentaires : ● si la machine fonctionne avec le système disque DOS 3.1 ou 3.2, se reporter au 7.10, pour plus détails. Néanmoins, le fonctionnement de l'instruction est le même dans les deux cas.

● *numéro de canal* est un numéro compris entre 0 et 7. Il y a affichage du message ?ILLEGAL QUANTITY ERROR si il est négatif ou supérieur à 255. *numéro de canal* peut être calculé par une expression arithmétique quelconque.

● si *numéro de canal* ou l'expression numérique qui le calcule vaut 0, toutes les entrées de données ou de programmes suivant cette instruction se feront à partir du *clavier* de la machine.

● si *numéro de canal* ou l'expression numérique qui le calcule est compris entre 1 et 7, les entrées de données ou de programmes suivant cette instruction se feront à partir du périphérique connecté au connecteur de *numéro de canal*. On rappelle ici que le canal 0 NE CORRESPOND PAS au connecteur 0 (sur lequel est souvent d'ailleurs connectée la carte APPLESOFT).

● si *numéro de canal* ou l'expression qui le calcule est supérieur à 7 (et inférieur à 256), ou si aucun périphérique ni contrôleur de périphérique n'est connecté au connecteur de *numéro de canal*, le système se place dans un état de suspension infinie qui ne peut être interrompu que par l'action de la touche RESET et le retour sous le contrôle de l'interpréteur BASIC APPLESOFT se fait par l'intermédiaire de *ctrl* C et RETURN.

Exemple :

10 IN # 1	<i>si par exemple est connecté</i>
20 INPUT A, B, C	<i>un deuxième clavier sur le</i>
30 IN # 0	<i>connecteur 1, entre les</i>
40 ...	<i>instructions 10 et 30 tous</i>
	<i>ordres INPUT seraient exécutés</i>
	<i>en prenant les données</i>
	<i>sur ce deuxième clavier.</i>

7.10 IN #

Objet : changer le numéro du canal par lequel s'effectuent les entrées vers la machine *sous le contrôle du système disque* DOS 3.1 ou DOS 3.2.

Syntaxe : IN # *numéro de canal*

Commentaires : ● *numéro de canal* est ici *nécessairement* un nombre entier compris entre 0 et 16. Si il est supérieur à 16 en mode d'exécution immédiat, et 7 en mode différé, un message d'erreur apparait : dans le premier cas RANGE ERROR dans le second SYNTAX ERROR (messages d'erreurs non précédés du point d'interrogation identifiant les erreurs détectées par l'interpréteur BASIC APPLESOFT).

- le fonctionnement de l'instruction est le même que pour l'instruction de même nom sans système disque :
- IN # 0 permet de faire toutes les communications *vers* la machine par l'intermédiaire du clavier de la machine.
- IN # *i* où *i* est le numéro (non nul) du connecteur sur lequel est connecté le périphérique par lequel les communications *vers* la machine vont être effectuées.
- *le connecteur 0 NE CORRESPOND PAS à un canal d'entrée/sortie accessible par l'instruction* IN.

● si *numéro de canal* est le numéro d'un connecteur dans le quel est placé un contrôleur de disquettes, le système disque est automatiquement rechargé avec donc comme conséquence la perte des programmes en mémoire.

● pour rester sous le contrôle du DOS en utilisant l'instruction IN # , il faut utiliser l'instruction DO IN # et non l'instruction BASIC APPLESOFT de même nom :

- en mode immédiat, la syntaxe étant la même (ou presque), il n'y a pas de différence sensible.
- en mode différé *par contre*, l'instruction BASIC APPLESOFT s'écrit IN = comme en mode immédiat ; l'instruction DOS au contraire utilise un ordre PRINT comme intermédiaire et l'on devra donc écrire :

● PRINT " *ctrl* D IN # 1 "

ctrl D précédé par un retour chariot indiquant que les caractères qui suivent jusqu'au prochain retour chariot sont des ordres DOS.

L'utilisation de l'ordre DOS à la place de l'ordre BASIC APPLESOFT identique permet de ne pas perdre le contrôle du DOS qui est partiellement déconnecté par l'ordre BASIC. (Pour en savoir plus sur le fonctionnement de l'instruction IN # sous DOS, il est conseillé de se reporter à la brochure DOS 3.2).

7.11 PR #

Objet : changer le numéro du canal par lequel la machine fournit les informations à l'extérieur, et par conséquent changer d'organe courant de sortie.

Syntaxe : PR # *numéro de canal*
exp num

Commentaires : ● *si la machine fonctionne sous le contrôle d'un système disque* DOS 3.1 ou DOS 3.2, *se reporter au 7.12. Néanmoins, le fonctionnement courant de l'opération est le même dans les deux cas.*

- *numéro de canal* ou *exp num* qui le calcule doit être compris entre 0 et 7. Il y a affichage du message ?ILLEGAL QUANTITY ERROR si il est négatif ou supérieur à 255.
- si *numéro de canal* ou *exp num* qui le calcule vaut 0, toutes les informations fournies par la machine vers l'extérieur apparaissent sur l'écran de la machine.

● si *numéro de canal* ou *exp num* qui le calcule est un nombre compris entre 1 et 7, les informations destinées à l'extérieur sont envoyées vers le connecteur de numéro *numéro de canal*. *On rappelle ici que le connecteur 0 NE CORRESPOND PAS au canal 0. Le connecteur 0 ne peut pas servir à des transferts d'informations dans le langage BASIC APPLESOFT.*

● si le numéro du canal d'entrée / sortie adressé par l'instruction PR # est compris entre 8 et 256 ou si aucun périphérique ni contrôleur de périphérique n'est connecté au connecteur de numéro *numéro de canal*, le système *risque* de se placer dans un état d'attente infinie lors du premier transfert d'informations vers l'extérieur. Dans ce cas, le seul moyen d'interrompre cette attente sans fin est d'actionner la touche RESET puis de faire *ctrl* C et RETURN.

Exemple :

10 PR # 1	<i>Supposons qu'une imprimante soit connectée au connecteur numéro 1. Les valeurs de</i>
20 PRINT A, B, C	<i>seront imprimées</i>
30 PR # 0	<i>sur cet appareil puis à partir de l'instruction 30 toutes les</i>
40 PRINT " FIN "	<i>impressions sont de nouveaux sur l'écran.</i>

7.12 PR #

Objet : changer le numéro de canal par lequel le système fournit les informations à l'extérieur *sous le contrôle du système disque* DOS 3.1 ou DOS 3.2.

Syntaxe : PR # *numéro de canal*

Commentaires : ● *numéro de canal* est ici *nécessairement* un nombre entier compris entre 0 et 16. Si en mode d'exécution immédiat, il dépasse 16, il y a affichage du message d'erreur RANGE ERROR. Si en mode d'exécution différé, il dépasse 8, il y a affichage du message SYNTAX ERROR. (On remarquera que ces deux message d'erreurs ne commencent pas par le point d'interrogation ? identifiant les erreurs détectées par l'interpréteur BASIC APPLESOFT car ce sont des erreurs détectées par le système disque DOS).

- le fonctionnement est le même que pour l'instruction BASIC APPLESOFT de même nom :
- PR # 0 permet d'envoyer sur l'écran de la machine toutes les informations destinées à l'extérieur.
- PR # *i* ou *i* est différent de 0 et identifie un connecteur sur lequel est connecté soit un contrôleur de périphérique de sortie soit un périphérique lui-même vers lequel la machine va envoyer les informations destinées à l'extérieur.
- *le connecteur 0 ne correspond pas à un canal d'entrée / sortie accessible par l'instruction* PR # .

● si *numéro de canal* est le numéro d'un connecteur sur lequel est placé un contrôleur de disquettes, l'instruction aura pour effet de recharger le système disque et donc de détruire le contenu actuel de la mémoire y compris les programmes.

● pour rester sous le contrôle du système disque en changeant l'assignation de l'organe courant de sortie, il faut utiliser l'instruction DO PR # et non l'instruction BASIC APPLESOFT de même nom :

- en mode immédiat, c'est toujours le cas car les syntaxes étant les mêmes, le DOS considère l'ordre PR # comme un ordre DOS ;
- en mode différé *par contre*, l'instruction BASIC APPLESOFT s'écrit comme en mode immédiat PR # ; l'ordre DOS de même nom nécessite l'emploi d'une instruction PRINT comme intermédiaire et l'on devra donc écrire :

● PRINT " *ctrl* D PR # 1 "

ctrl D précédé par un retour chariot indiquant que tous les caractères imprimés qui suivent jusqu'au prochain retour chariot sont des ordres DOS.

L'utilisation de l'ordre **DOS** à la place de l'ordre **BASIC APPLESOFT** identique permet de ne pas perdre le contrôle du **DOS** qui est partiellement déconnecté par l'ordre **BASIC APPLESOFT**.

8. COMMANDES.

Les commandes sont des ordres BASIC APPLESOFT qui permettent de contrôler le fonctionnement de la machine. Bien que ces ordres puissent être utilisés dans le corps d'un programme (à quelques exceptions près), ils sont généralement destinés à une exécution immédiate.

Ces commandes permettent :

- d'afficher et supprimer tout ou partie d'un programme en mémoire ;
- lancer l'exécution d'un programme en mémoire, assurer son interruption éventuelle et la reprise de son exécution après interruption ;
- de suivre à la " trace " le déroulement d'un programme pour détecter des erreurs éventuelles à des fins d'aide à la programmation et la mise au point ;
- de stocker sur cassette et charger en mémoire un programme ;
- d'initialiser la mémoire et les variables ;
- de quitter temporairement le BASIC APPLESOFT ou définitivement pour passer sous le contrôle d'un autre langage ou de programmes de l'utilisateur.

8.1 LIST

Objet : afficher tout ou partie du programme BASIC APPLESOFT actuellement en mémoire.

Syntaxe : LIST (*ligne 1*) ('- *ligne 2*)

Commentaires : ● si *ligne 1* et *ligne 2* sont présents tous deux, l'instruction afficher sur l'écran ou sur l'organe courant de sortie le contenu du programme compris entre les lignes *ligne 1* à *ligne 2*, cette dernière incluse. Si *ligne 1* ne correspond pas à un numéro de ligne existante, l'affichage commence à la première ligne existante de numéro supérieure à *ligne 1*. Si *ligne 2* ne correspond pas à un numéro de ligne existante, l'affichage s'interrompte à la dernière ligne de numéro inférieur à *ligne 2*. Si *ligne 1* est absent ou nul, l'affichage se fait à partir de *ligne 1* jusqu'à la fin du programme. Si aucun délimiteur - ou, ne suit *ligne 1*, seule cette ligne est affichée (si elle existe). Si *ligne 2* est inférieure à *ligne 1*, aucune ligne n'est affichée. En conséquence, les expressions suivantes affichent l'ensemble du programme :

```
LIST
LIST 0
LIST ,0
LIST -0
```

On prendra garde au fait que LIST, Q (par exemple) affiche l'ensemble du programme avant d'afficher le message ?SYNTAX ERROR.

● l'interpréteur **BASIC APPLESOFT** condense les programmes en codant les mots correspondant à des instructions ou des symboles particuliers. Lorsqu'on affiche un programme par la commande **LIST**, l'interpréteur décode ce programme condensé et insère des blancs entre les divers éléments qui constituent les instructions. Ainsi, le programme qui a été rentré au clavier ne ressemble-t-il pas exactement à l'image qu'on en obtient par l'instruction **LIST**. L'exemple le plus frappant est peut-être le suivant :

10?A

LIST 10

10 PRINT A

En effet, l'interpréteur BASIC APPLESOFT reconnaît comme forme valide du mot PRINT le point d'interrogation ; néanmoins, le code correspondant est celui de l'instruction PRINT et c'est donc ce mot qui apparaît dans l'affichage. Des blancs sont aussi ajoutés en fin de chaque ligne d'écran. Ceci peut être gênant quand ces espaces interviennent au milieu de chaînes de caractères (où les blancs sont significatifs) et qu'on veut modifier la ligne d'instruction en se servant des caractères affichés. Ceci peut être aussi gênant quand une instruction contient près du maximum autorisé de 239 caractères, et que, donc affichée avec des blancs, elle dépasse 239 caractères, nombre auquel elle sera tronquée si on utilise l'affichage pour la modifier. On dispose d'un moyen pour supprimer ces blancs non significatifs : il suffit en effet de fixer la longueur des lignes à un nombre inférieur à 34 caractères, ce qui est opéré par l'instruction suivante :

POKE 33, *nombre de caractères par ligne*

Ainsi dans l'exemple suivant :

10 C=A + B ligne d'instruction entrée au clavier

LIST *ordre d'afficher le programme*

10 C = A + B *résultat de l'affichage*

POKE 33,33 on fixe la longueur des lignes à 33 caractères

LIST *ordre d'afficher le programme*

10C = A + B *résultat de l'affichage maintenant condensé*

- LIST peut néanmoins être utilisé à l'intérieur d'un programme. Ceci peut être utile quand on dispose du système disque et qu'on désire créer un fichier contenant le *texte* du programme. Etant donné que les ordres d'écriture sur disquette ne peuvent être exécutés qu'en mode différé, il est nécessaire d'utiliser l'instruction LIST dans le même mode.

- l'action simultanée sur les touches CTRL et C interrompent l'affichage (sans possibilité de le reprendre par la commande CONT)

8.2 DEL

Objet : supprimer du programme en mémoire tout ou partie des lignes de ce programme.

Syntaxe : DEL *ligne 1, ligne 2*

Commentaires : ● DEL supprime du programme les lignes *ligne 1* à *ligne 2* incluse. Si la ligne de numéro *ligne 1* n'existe pas, la suppression commence à la première ligne existante de numéro supérieur à *ligne 1*. Si la ligne de numéro *ligne 2* n'existe pas, la suppression s'arrête à la dernière ligne existante de numéro inférieur à *ligne 2*.

- si la syntaxe normale n'est pas respectée, les indications qui suivent indiquent soit l'action effectuée, soit le message d'erreur en regard de la commande formulée.

COMMANDE ACTION ou MESSAGE

DEL
DEL , ?SYNTAX ERROR
DEL , *numéro de ligne*
DEL - *ligne 1 (, ligne 2)*

DEL 0, <i>ligne 2</i>	- supprime la ligne 0 si elle existe quelle que soit la valeur de <i>ligne 2</i> .
DEL 1, - <i>ligne 2</i>	- sans effet dans tous les cas

- DEL *ligne 1*, - *ligne 2*
- ?SYNTAX ERROR si *ligne 1* est plus grand que le numéro de la ligne le plus bas, sauf si ce dernier vaut 0 ou 1.
 - sans effet si l'unique ligne du programme a pour numéro 1 et *ligne 1* est non nul.
 - sans effet si *ligne 1* est non nul et inférieur ou égal à la ligne de numéro le bas.
- DEL *ligne 1* (,)
- sans effet dans tous les cas.
- DEL *ligne 1*, *ligne 2*
- sans effet si *ligne 1* est positif et supérieur à *ligne 2*.

● DEL peut être utilisé en mode d'exécution différé mais l'exécution du programme comprenant cette instruction cesse après la suppression de la dernière ligne. La commande CONT ne peut pas être utilisée pour continuer.

● la suppression d'une ou plusieurs lignes par DEL remet à zéro toutes les variables utilisées auparavant.

● pour supprimer une ligne unique, il suffit d'écrire le numéro de la ligne et de l'envoyer (c-a-d de frapper RETURN) : on remplace ainsi la ligne existante par une ligne sans instructions qui est donc ignorée par l'interpréteur BASIC APPLESOFT.

8.3 RUN

Objet : lancer l'exécution d'un programme soit à partir de l'instruction de numéro le plus bas, soit à partir du numéro de ligne indiqué.

Syntaxe : RUN (*numéro de ligne*)

Commentaires : ● RUN met à zéro toutes les variables, indicées ou non, les pointeurs (boucles FOR ... NEXT, sous programmes GOSUB, liste de données DATA, ...) et commence l'exécution du programme à la première instruction de la ligne de numéro *numéro de ligne* si ce paramètre est présent et à la première instruction du programme si RUN est employé sans paramètre.

● en mode d'exécution différé, l'utilisation de RUN met fin au programme courant pour exécuter les instructions à partir de *numéro de ligne* ou de la ligne la plus basse (comme ci-dessus). Si *numéro de ligne* est négatif ou si la ligne indiquée n'existe pas dans le programme courant, il y a affichage du message d'erreur ?UNDEF'D STATEMENT ERROR et arrêt du programme. Si *numéro de ligne* est supérieur à 63999, il apparaît le message ?SYNTAX ERROR et le programme s'arrête.

● En mode d'exécution immédiat, ces deux messages d'erreur apparaissent mais sont suivis d'une indication de numéro de ligne, généralement supérieur à 65 000 : ces numéros de ligne ne sont pas significatifs.

Les instructions d'un programme en mode immédiat qui suivent l'ordre RUN ne sont pas exécutées.

● si le système disque DOS 3.1 ou DOS 3.2 est utilisé, l'instruction RUN peut être utilisée pour exécuter un programme situé sur disquette en le chargeant en mémoire puis en l'exécutant à partir de l'instruction de numéro le plus bas. On rappelle ici la syntaxe de cette commande DOS, en priant le lecteur de se reporter à la brochure DOS correspondante pour plus d'informations.

RUN *nom de programme* (, S *numéro de canal*) (, D *numéro de drive*) (, V *numéro de volume*) où *nom de programme* est le nom d'un fichier contenant un programme écrit en BASIC APPLESOFT.

Exemple :

10 PRINT "10 "

20 PRINT "20 "

RUN *exécution de l'ensemble du programme*

10

20

RUN 20 *exécution du programme à partir de la ligne 20*

20

8.4 ctrl C

Objet : interrompre un programme BASIC APPLESOFT en cours d'exécution ou interrompre l'affichage sur l'écran d'un programme commandé par LIST.

Syntaxe : ctrl C

Commentaires : ● ctrl C permet d'interrompre l'exécution d'un programme à la fin de l'instruction en cours, ce qui est équivalent à la rencontre d'une instruction STOP à la suite de cette instruction en cours. Si aucune ligne de programme n'est modifiée ni supprimée, et si aucune erreur n'est apparue depuis l'interruption par ctrl C, l'exécution peut être reprise par l'instruction CONT.

● si ctrl C est utilisé pour interrompre un programme exécutant une instruction INPUT, ctrl C doit être suivi d'un retour chariot RETURN pour que l'interruption soit prise en compte. L'exécution du programme NE peut PAS être poursuivie par la commande CONT.

● si l'option ON ERR GOTO est utilisée dans le programme, ctrl C n'agit cependant pas de la même façon : l'interruption peut être traitée à l'intérieur du programme. ctrl C a pour effet de placer dans la zone de codes erreur la valeur 255 et cette valeur peut être donc testée pour savoir si l'on a affaire à une erreur ou une simple interruption. Pour reprendre l'exécution du programme là où ctrl C l'a interrompu, il suffit d'utiliser l'instruction RESUME. On se reportera néanmoins pour plus de détails au 6.8.

● si l'on utilise le système disque DOS 3.1 ou DOS 3.2, on prendra garde au fait que l'interruption créée par ctrl C peut troubler le fonctionnement des échanges avec le disque lorsqu'on reprend ensuite l'exécution du programme interrompu. ctrl C est donc à utiliser avec circonspection si l'option ON ERR GOTO n'est pas utilisée dans un fonctionnement avec un DOS.

8.5 CONT

Objet : poursuivre l'exécution d'un programme interrompu par ctrl C, STOP ou END.

Syntaxe : CONT

Commentaires : ● CONT permet de reprendre l'exécution d'un programme interrompu par ctrl C, STOP ou END à l'instruction suivant l'interruption, c'est à dire qu'aucune instruction du programme n'est sautée. Néanmoins, la reprise de l'exécution n'est pas toujours possible.

Elle n'est pas possible si l'interruption provient d'un ctrl C intervenu pendant l'exécution d'une instruction INPUT : la commande CONT a pour effet de provoquer le message d'erreur ?SYNTAX ERROR IN *numéro de ligne de l'ordre INPUT interrompu*.

Elle n'est pas possible si à la suite d'une interruption quelconque,

- une ou plusieurs lignes de programmes ont été modifiées ou supprimées ;

- une ou plusieurs erreurs sont intervenues depuis l'interruption. Néanmoins, les valeurs assignées aux diverses variables utilisées dans le programme peuvent être modifiées sans problèmes tant que ces modifications se font sans erreur.

- l'utilisation de CONT pour reprendre l'exécution d'un programme interrompu par la manipulation de la touche RESET n'est pas à conseiller et d'ailleurs est souvent impossible, un certain nombre de pointeurs ayant été altérés.

- si des erreurs sont intervenues ou si le contenu du programme a été modifié, l'utilisation de l'instruction CONT se traduit par l'affiche du message ?CANT CONTINUE ERROR (c'est à dire impossibilité de continuer).

- on rappelle enfin que l'utilisation de CONT n'est pas toujours compatible avec la poursuite d'un programme contenant des ordres d'écriture ou de lecture d'informations sur disquette avec les systèmes DOS 3.1 et DOS 3.2.

- si la commande CONT est utilisée dans un programme en mode différé (!), le programme s'arrête à cette instruction mais le contrôle de la machine n'est pas rendu à l'utilisateur qui doit utiliser *ctrl C* pour reprendre "la main". Mais l'utilisation de CONT dans ces circonstances transfèrera de nouveau le contrôle au programme interrompu, d'où de nouveau *ctrl C*, etc...

- si aucun programme n'a été interrompu, la commande CONT reste sans effet.

8.6 TRACE

Objet : aider à la mise au point d'un programme en affichant le numéro de la ligne de chaque instruction exécutée.

Syntaxe : TRACE

Commentaires : ● le numéro de la ligne contenant chaque instruction exécutée est affiché sur l'écran, précédé du symbole =. Comme aucun saut de ligne ni retour de chariot n'est effectué par TRACE lorsqu'il affiche un numéro de ligne, les affichages éventuels fournis par le programme exécuté se mélangent de façon assez peu claire avec les numéros de ligne.

- le mode de fonctionnement mis en œuvre par TRACE ne peut être abandonné pour le mode normal que par l'instruction NOTRACE ou par RESET *ctrl B* suivi de RETURN. Néanmoins cette dernière façon de procéder a l'inconvénient de vider tout programme de la mémoire !

- le mode TRACE ne peut être directement utilisé avec des programmes utilisant des ordres DOS, car TRACE ne fournit pas de retour chariot après chaque numéro de ligne et en conséquence la séquence RETURN *ctrl D* qui définit le début d'un ordre DOS ne peut plus être rencontrée. Pour pallier cet inconvénient, on peut remplacer *ctrl D* par la chaîne de caractères CHRS (13) + CHRS (4), c'est à dire la concaténation d'un retour chariot et d'un *ctrl D*.

8.7 NOTRACE

Objet : quitter le mode TRACE pour revenir au fonctionnement habituel.

Syntaxe : NOTRACE

Commentaires : ● si le mode TRACE n'a pas été activé par une instruction TRACE, la commande NOTRACE reste sans effet. Dans le cas contraire, elle rend à un programme BASIC APPLESOFT son fonctionnement habituel.

8.8 LOAD

Objet : charger un programme BASIC APPLESOFT inscrit sur cassette.

Syntaxe : LOAD

Commentaires : ● LOAD s'emploie sans paramètres. Les opérations effectuées par cette instruction sont les suivantes :

- détruire le programme actuellement en mémoire, mettre à zéro toutes les variables et les divers pointeurs ;

- émettre un signal sonore indiquant le début de l'opération de chargement proprement dite, le lecteur de cassette devant alors être en position de lecture et en fonctionnement ;

- charger le programme et émettre un signal sonore indiquant la fin du chargement. Le programme est alors chargé et prêt à l'emploi.

- l'utilisation de LOAD en mode différé se traduit par l'arrêt du programme contenant cette instruction après chargement éventuel d'un programme sur cassette.

- LOAD ne peut être interrompu que par l'action de la touche RESET. L'utilisation reprend le contrôle de la machine en BASIC en faisant *ctrl C* et RETURN.

- on prendra garde à ne pas utiliser de variables commençant par les lettres LOAD car la rencontre d'instructions du type LOADTRUC = 1 ont pour effet de suspendre l'exécution de tout programme, car l'interpréteur BASIC APPLESOFT détruit le programme courant puis tente de charger un programme à partir d'un lecteur de cassette absent ou pas prêt. Seul RESET peut interrompre cette attente.

- si on utilise le BASIC APPLESOFT en conjonction avec un système disque DOS 3.1 ou DOS 3.2, la commande DOS de même nom LOAD permet de charger des programmes situés sur disquette. Nous rappelons ici pour commodité la syntaxe de cette instruction DOS, et conseillons au lecteur de se reporter à la brochure DOS pour plus amples informations :

LOAD *nom de programme* (, S *numéro de canal*) (, D *numéro de drive*) (, V *numéro de volume*) où *nom de programme* est le nom d'un programme écrit en BASIC APPLESOFT. Les fonctions réalisées par cette instruction DOS sont les mêmes que l'instruction BASIC APPLESOFT pour ce qui de la remise à zéro des variables et des pointeurs. Le programme corant est aussi détruit.

8.9 SAVE

Objet : conserver sur cassette l'image d'un programme BASIC APPLESOFT se trouvant en mémoire.

Syntaxe : SAVE

Commentaires : ● SAVE permet de sauvegarder un programme en mémoire sur cassette. Les opérations réalisées sont les suivantes :

- émettre un signal sonore indiquant le début de l'enregistrement (le magnétophone à cassette doit être en fonctionnement et en position enregistrement) ;

- envoyer vers le magnétophone les signaux correspondant au codage du programme ;

- émettre un signal sonore indiquant la fin de l'opération.

- aucune altération du programme en mémoire ni des variables n'est à craindre d'une opération de sauvegarde par la commande SAVE qui peut être utilisée sans problèmes en mode différé.

- on prendra garde à ne pas employer de variables qui commencent par les lettres SAVE car la rencontre d'instructions du type SAVE A = 1 a pour effet de déclencher l'ensemble des opérations ci-dessus avant de faire apparaître le message ?SYNTAX ERROR.

● l'action sur la touche RESET permet seule d'interrompre une commande SAVE. Pour reprendre le contrôle de la machine en BASIC APPLESOFT, il faut ensuite faire *ctrl* C et RETURN.

● *la machine ne vérifie pas que l'enregistreur est marché pour envoyer l'ensemble des signaux correspondant au programme. Il est de la responsabilité de l'utilisateur de s'assurer que le magnétophone est dans les conditions requises :*

- *en position* enregistrement
- *en marche (défilement de la bande magnétique).*

● *si le langage BASIC APPLESOFT est utilisé avec un système disque DOS 3.1 ou DOS 3.2, la commande DOS de même nom peut être utilisée pour sayvegarder le programme courant sur disquette. Nous rappelons ici par commodité la syntaxe de cette instruction DOS, mais nous conseillons au lecteur de se reporter pour plus amples informations à la brochure DOS :*

SAVE *nom de programme* (, S *numéro de canal*) (, D *numéro de drive*) (, V *numéro de volume*) où *nom de programme* est le nom du fichier de la disquette qui contiendra le programme ainsi conservé.

8.10 NEW

Objet : initialiser la mémoire pour pouvoir composer un nouveau programme.

Syntaxe : NEW

Commentaires : ● NEW s'emploie sans paramètres. Il détruit le programme courant et le contenu de toutes les variables.

● NEW doit être utilisé pour éviter qu'on mélange le programme qu'on est en train d'écrire à un programme précédemment en mémoire.

● NEW peut être utilisé en mode d'exécution différé mais son emploi se traduit par l'arrêt immédiat du programme le contenant puisque ce dernier programme est détruit par l'instruction.

8.11 CLEAR

Objet : remettre à zéro les valeurs de toutes les variables, les différents pointeurs et contenus de piles.

Syntaxe : CLEAR

Commentaires : ● CLEAR s'emploie sans paramètres. Il effectue l'opération d'initialisation opérée par la commande RUN qui est en quelque sorte un CLEAR suivi d'un branchement à l'adresse indiquée.

● CLEAR peut être utilisé dans un programme en exécution différé mais il faut prendre garde que si la commande CLEAR intervient dans une boucle ou un sous-programme, la rencontre de l'ordre NEXT ou RETURN se traduira par une erreur, les piles d'adresses ayant été remises à zéro.

8.12 &

Objet : passer le contrôle de la machine à un programme en langage machine.

Syntaxe : *chaîne de caractères quelconques*

Commentaires : ● lorsque l'interpréteur BASIC APPLESOFT recontre une instruction commençant par *&*, il passe le contrôle de la machine à l'adresse *hexadécimale* 3F5. Les adresses à partir de 3F8 étant dédiées à un autre emploi, on peut placer en 3F5 une instruction de saut à une adresse de début de programme écrit en langage machine.

● la chaîne de caractères suivant le symbole *&* peut être exploitée par ce programme de l'utilisateur. En utilisation en mode immédiat, la ligne d'instruction qui vient d'être entrée se trouve dans la mémoire tampon dite "buffer d'entrée" commençant à l'adresse *hexadécimale* 200. Le traitement de la chaîne de caractères suivant *&* peut donc être fait facilement à partir des données situées entre 200 et 300 (en *hexadécimal*). Ceci peut être commode pour pouvoir disposer à la fois du contrôle de la machine sous le langage BASIC APPLESOFT et sous le contrôle de ses programmes propres. En particulier, si la dernière instruction du programme en langage machine est une instruction de branchement à l'adresse 3D0 *hexadécimal*, le contrôle est rendu au BASIC APPLESOFT.

Exemple :

Supposons qu'on veuille accéder au mini-assembleur qui est disponible en mémoire morte et dont le point d'entrée est l'adresse *hexadécimale* F666. On place donc l'instruction de branchement à cette adresse soit JMP F666 en 3F5. Ainsi lorsqu'on frappera la touche *&*, on se trouvera sous le contrôle du mini-assembleur qui répond par un point d'interrogation '.

9. INSTRUCTIONS DE MISE EN PAGE DES AFFICHAGES.

Outre les différents paramètres de l'instruction PRINT décrits en 7.3, le BASIC APPLESOFT propose un jeu d'instruction permettant de déplacer le curseur, d'inverser le type d'affichage (noir sur blanc), de faire clignoter tout ou partie de l'écran, d'effacer l'écran et de faire varier la vitesse d'affichage.

9.1 HTAB

Objet : déplacer le curseur à une position donnée par rapport à la marge gauche.

Syntaxe : HTAB (*exp num*)

Commentaires : ● une ligne *logique* est constituée de 255 positions numérotées de 1 à 255. Quelle que soit la taille de la fenêtre définie pour les impressions, les positions 1 à 40 sont situées sur la ligne courante, les positions 41 à 80 sur la ligne suivante, etc... *exp num* permet d'évaluer la position à laquelle va être placé le curseur par l'action de l'instruction HTAB. Ainsi HTAB (1) place le curseur au début de la ligne courante, HTAB (41) le place au début de la ligne suivante, etc... Si la largeur de la fenêtre d'affichage fait moins de 40 caractères de large, HTAB peut déplacer le curseur à l'extérieur de la fenêtre d'affichage mais un seul caractère pourra être ainsi placé à l'extérieur de la fenêtre par un ordre PRINT, le suivant passant à la ligne suivante.

- on notera que HTAB (0) place le curseur à la 256^{ième} position de la ligne courante.
- *exp num* doit prendre des valeurs entre 0 et 255. A l'extérieur de ce domaine, il y a affichage du message ?ILLEGAL QUANTITY ERROR.

- on notera les différences entre TAB et HTAB :
 - TAB doit être utilisée dans un ordre PRINT ; HTAB est une instruction.
 - TAB ne permet pas de faire reculer le curseur : si l'argument de la fonction TAB est inférieur à la position courante du curseur, ce dernier n'est pas déplacé. HTAB permet au contraire de replacer le curseur au début la ligne par exemple.

- TAB déplace le curseur à l'intérieur de la fenêtre d'affichage et passe à la ligne suivante pour continuer la tabulation si la limite droite de la fenêtre est franchie. HTAB au contraire ne passe à la ligne suivante que lorsque le curseur atteint les positions 40, 80, ... et ne tient pas compte de la fenêtre définie.

9.2 VTAB

Objet : déplacer le curseur sur une ligne donnée par rapport au sommet de l'écran.

Syntaxe : VTAB (*exp num*)

Commentaires : ● l'écran est constitué de 24 lignes numérotées de 1 à 24, la ligne 1 correspondant à la plus haute ligne. *exp num* permet d'évaluer le numéro de la ligne à laquelle va être déplacé le curseur par l'action de VTAB. Ce déplacement est uniquement vertical et ne tient pas compte des limites de la fenêtre d'affichage. Les déplacements sont absolus c'est à dire que VTAB (1) place toujours le curseur sur la ligne la plus haute de l'écran et VTAB (24) sur la ligne la plus basse.

● *en mode graphique*, VTAB déplace le curseur dans la zone graphique de l'écran ce qui permet de commenter les graphiques.

● si VTAB déplace le curseur à l'extérieur de la fenêtre d'affichage, tous les affichages suivants se feront sur cette ligne.

● *exp num* doit prendre une valeur comprise entre 1 et 24 sinon il y a affichage du message ?ILLEGAL QUANTITY ERROR.

9.3 HOME

Objet : effacer l'écran à l'intérieur de la fenêtre d'affichage et placer le curseur au début de la première ligne de la fenêtre d'affichage.

Syntaxe : HOME

Commentaires : ● HOME s'emploie pour effacer tous les caractères contenus à l'intérieur de la fenêtre d'affichage et repositionner le curseur dans le coin haut gauche de cette fenêtre.

● HOME est identique à l'appel du sous-programme du moniteur situé en 64 000, soit CALL - 936, ou encore à l'action au clavier des touches ESC et a.

9.4 FLASH

Objet : faire clignoter tous les caractères affichés par un ordre d'impression.

Syntaxe : FLASH

Commentaires : ● L'instruction FLASH permet de faire clignoter tous les caractères affichés par un ordre d'impression suivant l'établissement de ce mode de visualisation. C'est à dire que les caractères affichés avant l'établissement de ce mode ne sont affectés par cette instruction ni les entrées au clavier affichées par l'ordre INPUT (par exemple). Le clignotement consiste à faire apparaitre les caractères tantôt en blanc sur fond noir, tantôt en noir sur fond blanc.

● le mode d'affichage normal est rétabli par l'instruction NORMAL, mais les caractères clignotant le restent.

Exemple :

```
10 PRINT " PROGRAMME "  le mot programme est affiché normalement ;

20 FLASH                à partir de maintenant et jusqu'à la rencontre de l'
                        instruction NORMAL, les caractères imprimés sont clignotants.

30 PRINT " TEST "

                        les caractères sont de nouveau imprimés normalement.

40 NORMAL

50 PRINT " NO 1 "
```

9.5 INVERSE

Objet : inverser le mode d'affichage : les caractères sont affichés en noir sur fond blanc.

Syntaxe : INVERSE

Commentaires : ● l'inversion de mode d'affichage n'affecte que les caractères affichés par un ordre d'impression (normalement PRINT) après l'établissement de ce mode. C'est à dire que les caractères affichés en réponse à un ordre INPUT (par exemple) seront affichés en mode normal, de même que les caractères déjà affichés ne sont pas affectés par l'instruction INVERSE.

● l'instruction NORMAL permet de rétablir le mode d'affichage normal mais les caractères affichés en mode inverse le restent.

9.6 NORMAL

Objet : rétablir le mode d'affichage vidéo normal.

Syntaxe : NORMAL

Commentaires : ● NORMAL rétablit l'affichage normal, c'est à dire caractères blancs sur fond noir, pour les caractères affichés aussi bien par un ordre d'impression (par exemple PRINT) que par un ordre d'entrée visualisant les caractères fournis (par exemple INPUT).

● NORMAL rétablit donc en particulier les modes FLASH et INVERSE établis par les instruction de même nom.

9.7 SPEED

Objet : fixer la vitesse de sortie des données sur l'écran ou un périphérique quelconque.

Syntaxe : SPEED = *exp num*

Commentaires : ● SPEED affecte le débit d'informations des instructions de sortie sur l'écran et tout autre canal de sortie (sauf le disque s'il est présent). *exp num* doit être comprise entre 0 et 255 pour éviter le message d'erreur ?ILLEGAL QUANTITY ERROR.

● La valeur par défaut donnée à la vitesse d'affichage est 255. On a mesuré les valeurs suivantes pour le débit d'informations :

SPEED = 0	6,15 caractères par seconde
SPEED = 100	15,67 caractères par seconde
SPEED = 255	1305 caractères par seconde

10 LES INSTRUCTIONS DE GRAPHIQUES ET DE JEUX.

10.1 TEXT

Objet : passage du mode graphique au mode texte.

Syntaxe : TEXT

Commentaires : ● cette instruction provoque le passage du mode graphique haute ou basse résolution au mode texte (40 caractères par ligne, 24 lignes) et place le curseur à la dernière ligne de l'écran. En mode texte, l'instruction TEXT est équivalente à VTAB 24.

TEXT annule les commandes spéciales de formatage d'écran.

10.2 GR

Objet : provoque l’affichage en mode graphique basse résolution.

Syntaxe : GR

Commentaires : ● l’écran est alors effacé et comprend une partie graphique (40 × 40) et 4 lignes en mode texte au bas de l’écran. Le curseur se trouve au début du texte.

● POKE-16302,0 (ou POKE 49234,0) permet d’utiliser tout l’écran en basse résolution (40 × 48). Après la commande GR, la couleur est mise à 0.

● GR peut être employé après le mode HGR. Mais après le mode HGR2, elle laisse l’utilisateur en page 2 de la basse résolution.

Il est préférable d’utiliser la séquence TEXT, puis GR pour passer de la page 2 haute résolution à la page 1 basse résolution.

10.3 COLOR

Objet : fixer une couleur en basse résolution

Syntaxe : COLOR *exp num*

Commentaires : ● *exp num* peut être un nombre réel (il est alors converti en entier) et ses valeurs peuvent aller de 0 à 25 (elles sont traitées modulo 16).

● Liste des numéros et noms de couleur :

0 noir	4 vert foncé	8 marron	12 VERT
1 magenta	5 gris	9 orange	13 jaune
2 bleu foncé	6 bleu moyen	10 gris	14 bleu clair
3 violet	7 bleu brillant	11 rose	15 blanc

10.4 PLOT

Objet : placer un point en basse résolution.

Syntaxe : PLOT *exp num 1, exp num 2*

Commentaires : ● PLOT place un point de coordonnée x *exp num 1* et de coordonnée y *exp num 2* en basse résolution dans la dernière couleur indiquée par COLOR (couleur de numéro 0 si aucune autre n’a été précisée). *exp num 1* peut varier de 0 à 39 et *exp num 2* de 0 à 47, sinon le message ‘ ? ILLEGAL QUANTITY ERROR ’ apparaît. L’origine (0,0) se trouve au coin supérieur gauche, comme pour tout mode graphique.

10.4 HLIN

Objet : tracer un trait horizontal en basse résolution

Syntaxe : HLIN *exp num 1, exp num 2*

AT *exp num 3*

Commentaires : ● en basse résolution, HLIN trace un trait horizontal du point de coordonnées *exp num 2, exp num 3*.

● la couleur du trait est la dernière indiquée par COLOR. *exp num 1* et *exp num 2* peuvent aller de 0 à 39, *exp num 3* de 0 à 47, sinon le message ‘ ? ILLEGAL QUANTITY ERROR ’ apparaît. *exp num 1* peut être inférieure, supérieure ou égale à *exp num 2*.

10.5 VLIN

Objet : tracer un trait vertical en basse résolution

Syntaxe : VLIN *exp num 1, exp num 2*

AT *exp num 3*

Commentaires : ● de même nature que HLIN, VLIN trace un trait vertical du point de coordonnées *exp num 1, exp num 3* au point de coordonnées *exp num 2, exp num 3*. *exp num 1* et *exp num 2* vont de 0 à 47 et *exp num 3* de 0 à 39.

10.6 SCR N

Objet : connaître la couleur d’un point de l’écran en basse résolution.

Syntaxe : SCR N (*exp num 1, exp num 2*)

Commentaires : ● cette fonction donne le numéro de la couleur du point de coordonnées *exp num 1, exp num 2* en basse résolution.

10.7 HGR

Objet : provoque le passage en haute résolution page 1

Syntaxe : HGR

Commentaires : ● l’écran est alors effacé et l’affichage est en mode haute résolution (280 × 160) en laissant 4 lignes de texte en bas de l’écran. La valeur de HCOLOR n’est pas modifiée, la page texte est toujours présente et on en voit les 4 dernières lignes. Le curseur peut donc se trouver à n’importe quel endroit et n’apparaît que si on le place dans ces 4 lignes. POKE - 16302,0 (ou POKE - 49234,0) permet d’utiliser toute la page haute résolution (280 × 192). La page 1 haute résolution prend 8K de mémoire des adresses 8192 à 16384.

10.8 HGR2

Objet : provoque le passage en haute résolution page 2

Syntaxe : HGR2

Commentaires : ● la haute résolution page 2 (280 × 192) prend également 8K de mémoire des adresses 16384 à 2»576 sans possibilité de lignes texte. L’écran est effacé, la page texte n’est pas modifiée. L’avantage est de laisser plus de place pour les programmes et les variables.

10.8 HCOLOR

Objet : fixer une couleur en haute résolution.

Syntaxe : HCOLOR *exp num*

Commentaires : ● *exp num* doit prendre les valeurs de 0 à 7.
 ● liste des numéros et noms de couleurs.

0 noir	4 noir 2
1 vert	5 dépend de l'écran
2 bleu	6 dépend de l'écran
3 blanc 1	7 blanc

10.9 HLOT

Objet : tracer un point ou un trait en haute résolution.

Syntaxe : HLOT *exp num 1, exp num 2*

HLOT TO *exp num 1, exp num 2*

HLOT *exp num 1, exp num 2 TO exp num 3, exp num 4 (TO...)*

Commentaires : ● dans la première option, HLOT trace un point de coordonnées x *exp num 1* et de coordonnées y *exp num 2*.

 ● dans la deuxième option, HLOT relie le dernier point de coordonnées x *exp num 1* et de coordonnées y *exp num 2*.

 ● la troisième option permet de tracer une ligne d'un point de coordonnées *exp num 1, exp num 2* au point de coordonnées *exp num 3, exp num 4* et continuer en ajoutant TO *exp num 5, exp num 6...*

 ● exemple : tracé d'un rectangle en une seule instruction : HLOT 0,0 TO 279,0 TO 279,159 TO 0,0.

 ● dans le trois cas *exp num 1* et *exp num 3* peuvent aller de 0 à 279, *exp num 2* et *exp num 4* de 0 à 191. La couleur de ces différents tracés est celle désignée par la dernière instruction HCOLOR, sauf pour la deuxième option dont la couleur est celle du dernier point tracé, même si une couleur différente a été sélectionnée entre temps.

10.10 PDL

Objet : lire les valeurs des manettes de jeu.

Syntaxe : PDL (*exp num*)

Commentaires : ● cette fonction retourne une valeur comprise entre 0 et 255, correspondant à la position de la manette de jeu désignée par *exp num*. Il est nécessaire de séparer les instructions de lecture des manettes pour éviter certaines confusions dans les valeurs transmises.

 ● Les instructions X=PEEK (-16287), X=PEEK (-16286) et X=PEEK (-16285) permettent de savoir si les boutons des manettes de jeu 0, 1 et 2 ont été poussés. X est alors supérieur à 127. Beaucoup utilisée dans les jeux, l'instruction X=PEEK (-16336) provoque un petit beep par le haut parleur.

11 FONCTIONS MATHEMATIQUES ET TRAITEMENT DES CHAINES ALPHANUMERIQUES

- SIN (*exp num*) : Sinus de *exp num* en radian
- COS (*exp num*) : Cosinus de *exp num* en radian
- TAN (*exp num*) : Tangente de *exp num* en radian
- ATN (*exp num*) : Arctangente de *exp num* en radian
- INT (*exp num*) : Partie entière de *exp num*
- RND (*exp num*) : Nombre aléatoire compris entre 0 et 1
- SGN (*exp num*) : Signe de *exp num* exprimé par - 1, 0, et 1
- ABS (*exp num*) : Valeur absolue de *exp num*
- SQR (*exp num*) : Racine carrée de *exp num*
- EXP (*exp num*) : Exponentielle de *exp num*
- LOG (*exp num*) : Logarithme de *exp num*

- STR (*exp num*) : Conversion d'un nombre en chaîne alphanumérique
- LEN (*exp alpha*) : Donne la longueur d'une chaîne alphanumérique
- VAL (*exp alpha*) : Conversion d'une chaîne alphanumérique en nombre
- CHR (*exp num*) : Donne le caractère ASCII correspondant à *exp num*
- ASC (*exp alpha*) : Donne le code ASCII du caractère *exp alpha*
- LEFT (*exp alpha, exp num*) : Partie gauche, de longueur *exp num*, de *exp alpha*
- RIGHT (*exp alpha, exp num*) : Partie droite, de longueur *exp num*, de *exp alpha*
- MID (*exp apha, exp num 1, exp num 2*) : Partie de *exp alpha*, qui débute à *exp num 1* et qui a une longueur *exp num 2*

sonotec

5, RUE FRANÇOIS PONSARD 75016 PARIS

TÉLÉPHONE 524.37.40 (lignes groupées)

Télex Semoule Paris 610942