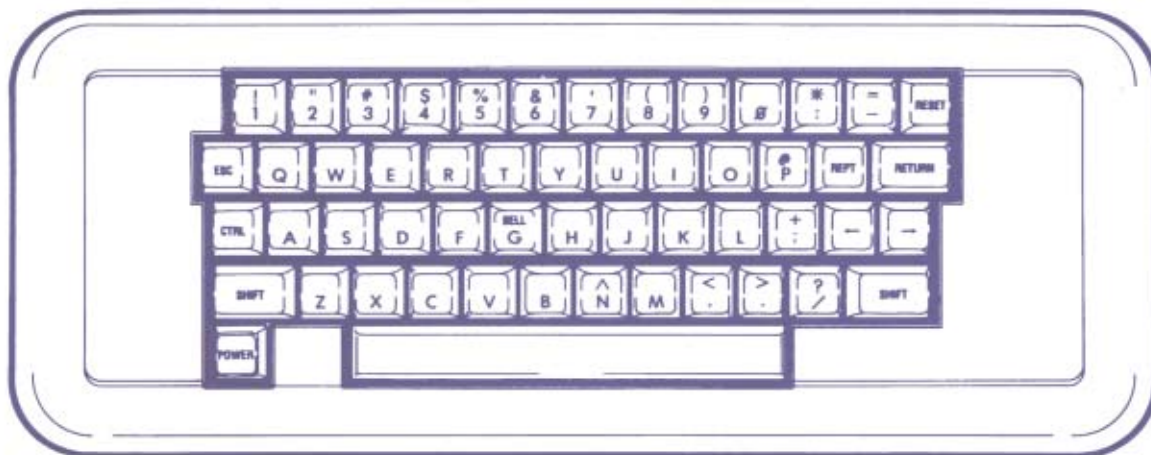

MICRO-ORDINATEUR
ITT 2020*
MANUEL D'UTILISATION
2eme partie :
BASIC ÉTENDU A VIRGULE FLOTTANTE (PALSOFT)

* Apple System



- Afin de faciliter la compréhension de ce manuel, les textes ont été composés dans deux caractères différents:
l'un pour les textes généraux, l'autre pour faire ressortir les textes qui doivent apparaître sur l'écran.

Exemple:

```
714 YSPEED = -1 * YSPEED
```

```
719 GOTO 706
```

qui apparaîtra sur l'écran de la façon suivante:

```
714 YSPEED=-1*YSPEED
```

```
719 GOTO 706
```

Notez bien la façon de différencier la lettre O du chiffre 0.

- Tout au long de ce manuel, vous verrez des indications de ce genre:

RETURN ou REPT ou encore "tapez 130 puis RETURN".
B

Cela signifie que vous devez appuyer sur la ou les touches dont les noms sont indiqués dans le ou les cartouches.

Dans l'exemple REPT il faut appuyer sur les deux touches simulta-

B
nément.

Traduit de l'américain par J.M. PIRONNEAU
"BASIC PROGRAMMING REFERENCE MANUAL" APPLESOFT II
édité par:

APPLE COMPUTER INC.
Cupertino, Cal. USA

pour le compte de:

ITT OCEANIC
97 avenue de Verdun - 93230 Romainville

1er trimestre 1980
IF Publicité & communication
Imprimé en France
R.C. Paris B 300960614 ,

Table des Matières

AVANT-PROPOS		page 10
CHAPITRE 1	"première approche"	page 12
	- commandes en mode immédiat	page 12
	- commandes en mode programme	page 12
	- formats des nombres	page 14
	- exemple de graphisme couleur	page 15
	- formats d'affichage	page 16
	- noms des variables	page 17
	- IF...THEN: tests	page 19
	- un autre exemple de graphisme couleur	page 20
	- FOR...NEXT: itération	page 21
	- les tableaux	page 24
	- GOSUB...RETURN: sous-programmes	page 25
	- READ...DATA...RESTORE: le traitement des données	page 26
	- variables réelles, entières et alphanumériques	page 27
	- chaînes de caractères	page 28
	- encore plus de graphisme couleur	page 32
	- graphisme couleur haute résolution	page 33
CHAPITRE 2	définitions	page 37
	- définitions syntaxiques et abréviations	page 37
	- règles de priorité dans les expressions	page 42
	- conversions des expressions	page 42
	- modes d'exécution	page 43
CHAPITRE 3	commandes du système et des utilitaires	page 44
	- LOAD, SAVE	page 44
	- NEW	page 44
	- RUN	page 44
	- STOP, END, CTRL C, RESET, CONT	page 45
	- TRACE, NOTRACE	page 46

- PEEK	page 46
- POKE	page 47
- WAIT	pages 47-48
- CALL	page 49
- HIMEM:	page 49
- LOMEM:	page 50
- USR	pages 50-51

CHAPITRE 4

commandes d'édition et de formats d'affichage	page 53
--	---------

(voir aussi CTRL C au chapitre 3)

- LIST	page 53
- DEL	page 54
- REM	page 55
- VTAB	page 55
- HTAB	page 55
- TAB	page 56
- POS	page 56
- SPC	page 56
- HOME	page 57
- CLEAR	page 57
- FRE	page 57
- FLASH, INVERSE, NORMAL	page 58
- SPEED	page 58
- esc A, esc B, esc C, esc D	page 59
- répétition	page 59
- flèche à droite, flèche à gauche	page 59
- CTRL X	page 60

CHAPITRE 5

tableaux et chaînes de caractères	page 61
-----------------------------------	---------

- DIM	page 61
- LEN	page 62
- STR\$	page 62

- VAL	page 62
- CHR\$	page 62
- ASC	page 63
- LEFT\$	page 63
- RIGHT\$	page 63
- MID\$	page 64
- STORE, RECALL	pages 64-65-66-67

CHAPITRE 6

commandes d'entrées/sorties	page 68
-----------------------------	---------

(voir aussi: LOAD, SAVE au chapitre 3
STORE, RECALL au chapitre 5)

- INPUT	page 68
- GET	page 69
- DATA	page 70
- READ	page 71
- RESTORE	page 72
- PRINT	pages 72-73
- IN #	page 73
- PR #	page 73
- LET	page 74
- DEF FN	pages 74-75

CHAPITRE 7

commandes relatives aux branchements	page 77
- GOTO	page 77
- IF...THEN, IF...GOTO	pages 77-78
- FOR...TO...STEP	page 78
- NEXT	pages 79-80
- GOSUB	page 80
- RETURN	page 80
- POP	page 81
- ON...GOTO, ON...GOSUB	page 81
- ONERR GOTO	pages 81-82
- RESUME	page 83

CHAPITRE 8

graphisme (et leviers de commandes)	page 85
-------------------------------------	---------

- TEXT	page 85
- graphiques couleurs large résolution	page 85
GR	page 85
COLOR	page 86
PLOT	page 86
HLIN	page 87
VLIN	page 87
SCRN	pages 87-88
- graphiques couleurs haute résolution	
HGR	page 88
HGR 2	page 89
HCOLOR	page 90
HPLOT	page 90
- leviers de commandes	
PDL	page 91

CHAPITRE 9

les figures haute résolution	page 93
- comment créer la table de construction des figures	pages 93 à 99
- conserver la table de construction des figures	page 100
- utiliser la table de construction des figures	pages 100-101
- DRAW	page 101
- XDRAW	page 102
- ROT	page 102
- SCALE	page 102
- SHLOAD	page 103

CHAPITRE 10

les fonctions mathématiques disponibles	page 105
- SIN, COS, TAN, ATN, INT, RND, SGN, ABS,	page 105
SQR, EXP, LOG	page 106

ANNEXES

ANNEXE A	mise en route	page 107
ANNEXE B	l'édition des programmes	page 111
ANNEXE C	messages d'erreurs	page 116
ANNEXE D	comment gagner de la place mémoire	page 119
ANNEXE E	pour accélérer la vitesse d'exécution de vos programmes	page 122
ANNEXE F	valeur décimale des mots-clés	page 124
ANNEXE G	mots réservés dans l'interpréteur du BASIC Etendu Virgule Flottante	page 126
ANNEXE H	pour convertir un programme BASIC en BASIC Etendu Virgule Flottante	page 129
ANNEXE I	organisation de la mémoire	page 131
ANNEXE J	instructions PEEK, POKE, CALL tables des variables et tableaux	page 134
ANNEXE K	codes ASCII des caractères	page 144
ANNEXE L	l'organisation de la page zéro avec le BASIC Etendu Virgule Flottante	page 150
ANNEXE M	différences entre BASIC Etendu Virgule Flottante et BASIC Entier	page 153
ANNEXE N	glossaire alphabétique des définitions syntaxiques et abréviations	page 156
ANNEXE O	résumé des commandes	page 162
ANNEXE P	index général	page 174
ANNEXE Q	index des commandes	page 187



AVANT-PROPOS

INTRODUCTION

Le BASIC Etendu à Virgule Flottante (Palsoft) est le langage BASIC évolué utilisé par l'ITT 2020. Le BASIC a été étendu parce que l'ITT 2020 possède des fonctions spécialisées qui ne sont pas disponibles sur d'autres ordinateurs utilisant le BASIC. L'ajout de quelques mots nouveaux au BASIC permet à tout utilisateur de l'ITT 2020 d'exploiter immédiatement ces fonctions spécialisées; parmi ceux-ci, vous trouverez le graphisme couleur en large et haute résolution, les figures et les leviers de commandes.

Ce manuel expose les caractéristiques particulières du B.E.V.F. Ce n'est pas un manuel d'initiation à la programmation, puisque ITT fournit pour cela un manuel d'utilisation séparé (le BASIC Entier).

Ce manuel suppose que vous connaissiez le BASIC et que vous désiriez connaître les nouvelles possibilités qu'offre le B.E.V.F.

Le chapitre 1 ("première approche") est un rapide résumé des possibilités offertes par le langage. La suite du manuel décrit soigneusement et complètement la formulation, la syntaxe et le fonctionnement de chaque instruction.

Afin d'éviter la déception et la contrariété que certains manuels peuvent causer, celui-ci indique les moments où des erreurs de programmation peuvent vous provoquer certaines difficultés, des symboles spéciaux attirent votre attention sur ces points.

La méthode utilisée pour décrire le B.E.V.F. est un langage assez simple en lui-même, vous vous apercevrez qu'après un court moment d'adaptation, il permettra d'accélérer votre compréhension de ce qui est, à proprement parler, légal et illégal dans ce langage. Vous n'aurez pas à affronter d'incessantes ambiguïtés concernant l'interprétation d'une phrase, comme cela peut arriver dans une description littéraire d'un sujet technique.

Les programmeurs confirmés trouveront ce manuel particulièrement utile. Les programmeurs débutants doivent se rappeler qu'ils ne seront bientôt plus débutants et apprécieront l'effort important que nous avons fait pour fournir un manuel extrêmement complet.

UTILISATION DU MANUEL

Ce manuel suppose que vous ayez un minimum de connaissances sur le langage BASIC. Si vous n'êtes pas familiarisé avec celui-ci, le manuel du BASIC Entier vous fournira une introduction à ce nouveau langage puisque ces deux BASIC ont de nombreux points communs.

Nous vous recommandons de charger en mémoire le B.E.V.F. quand vous consulterez ce manuel, afin de pouvoir essayer sur votre ITT 2020 tout ce que ce manuel explique et suggère. Le caractère de reconnaissance du B.E.V.F. (J) qui s'affichera sur l'écran vous indique que vous travaillez en BASIC Etendu à Virgule Flottante.

Référez-vous à l'annexe A pour savoir comment charger le BASIC Etendu à Virgule Flottante (B.E.V.F.) dans votre ordinateur. Il y a deux termes que vous devez connaître en abordant ce manuel. Le mot "syntaxe" fait référence à la structure d'une commande ou d'une instruction de l'ordinateur. Le verbe "interpréter" se réfère à la manière dont l'ordinateur essaye d'analyser ce que vous avez tapé, en distinguant les différentes parties de

l'instruction pour pouvoir l'interpréter. Par exemple la syntaxe du B.E.V.F. vous permet d'écrire $12 \times 5 = 4 * 3 \wedge 2$.

Quand le B.E.V.F. interprète ces informations, il distingue d'abord 12 comme numéro de ligne du programme, puis considère X5 comme un nom de variable arithmétique, puis évalue $3 \wedge 2$ à 9, le multiplie à 4 et assigne le total (36) à la variable dont le nom est X5.

Le chapitre 1 est une vue d'ensemble de nombreuses commandes du B.E.V.F., pour ceux qui ont peu d'expérience de la programmation en BASIC. Il introduit des concepts initiaux, avec des exemples commentés que vous pourrez essayer sur votre ordinateur.

L'annexe B donne des "tuyaux" pour éditer les programmes en B.E.V.F.

La notation introduite au début du chapitre 2 sert à décrire la syntaxe du B.E.V.F. de manière concise et sans ambiguïté. Elle vous permettra d'épargner votre temps pour savoir comment les instructions doivent être structurées dans leur construction.

Vous n'avez pas besoin d'utiliser cette notation pour vous-même, mais elle vous aidera à répondre à certaines questions qui ne sont pas particulièrement abordées dans ce manuel. Par exemple, les crochets ([,]) indiquent le caractère facultatif d'une partie d'instruction. Les accolades ({,}) servent à indiquer le caractère répétitif d'une partie d'instruction.

Par exemple:

[LET] C = 3 indique que le mot LET est facultatif et peut être omis.

REM [} caractère} indique que les REMarques sont constituées du mot REM éventuellement suivi d'un ou plusieurs caractères.

Les abréviations et définitions syntaxiques du chapitre 2 sont présentées dans un ordre logique pour ceux qui veulent savoir comment nous avons construit ce système de symboles et de définitions.

Si vous préférez ignorer cette construction vous pouvez vous référer au glossaire alphabétique des termes syntaxiques de l'annexe N, quand vous en aurez besoin dans le manuel.

Les chapitres 3 à 10 donnent des explications détaillées sur toutes les instructions du B.E.V.F. groupées par fonctions. Si vous recherchez précisément une instruction, un index alphabétique des instructions se trouve en dernière page de ce manuel.

Enfin, beaucoup de documentation complémentaire est contenue dans les annexes de la fin du manuel.



Précédant un paragraphe, indique une caractéristique inhabituelle qui doit attirer votre attention.



Précédant un paragraphe, décrit des situations où vous risquez d'effacer le B.E.V.F. de la mémoire et de perdre votre programme. Vous auriez alors à recommencer depuis le début

CHAPITRE 1

PREMIERE APPROCHE

COMMANDES EN MODE IMMÉDIAT

Tapez au clavier le texte suivant:

```
PRINT 10 - 4
```

puis appuyez sur la touche marquée **RETURN**.

B.E.V.F. affichera immédiatement 6 sur l'écran.

L'instruction PRINT que vous avez introduite s'est exécutée aussitôt que la touche a été enfoncée.

B.E.V.F., après avoir calculé l'opération qui suivait PRINT, a affiché 6, le résultat.

Maintenant, introduisez ceci au clavier:

```
PRINT 1/2, 3 * 10
```

(* est l'opération multiplication et / celui de la division).

Après avoir enfoncé la touche **RETURN**, B.E.V.F. affiche:

```
.5          30
```

Comme vous le constatez, B.E.V.F. effectue les divisions, multiplications, soustractions et additions. Notez que la virgule (,) intercalée entre les opérations de la commande PRINT a permis l'affichage de deux valeurs en simultané. L'utilisation des (,) dans les PRINT divise les lignes de 40 caractères en trois colonnes ou "champs de tabulation". Reportez-vous au paragraphe sur les champs de tabulation décrit avec la commande PRINT au chapitre 6.

COMMANDES EN MODE PROGRAMME

Les commandes telles la formulation "PRINT" que vous venez d'utiliser sont appelées des commandes en mode immédiat.

Il y a un autre type de commandes, appelées commandes en mode programme. Pour exécuter une commande en mode programme, la commande doit commencer par un numéro de ligne, c'est-à-dire un nombre entier compris entre 0 et 63999.

Tapez au clavier les lignes suivantes:

```
10 PRINT 2 + 3
20 PRINT 2 - 3
```

(N'oubliez pas qu'une ligne doit être suivie d'un retour de chariot, c'est-à-dire en appuyant sur la touche **RETURN**).

Une séquence de commandes (avec n° de ligne) s'appelle tout simplement un programme. Au lieu de s'exécuter immédiatement, les commandes sont stockées dans la mémoire de votre ITT 2020.

Et après avoir tapé RUN puis appuyé sur RETURN, les instructions s'exécutent, d'abord l'instruction ayant le plus petit numéro de ligne, puis toutes les suivantes, séquentiellement, dans l'ordre croissant des numéros de lignes, jusqu'à ce que le programme soit

entièrement exécuté.

Supposons que vous tapiez RUN maintenant (N'oubliez pas que `RETURN` doit être pressé à chaque fin de ligne), le B.E.V.F. affichera sur votre écran TV:

```
5  
-1
```

Dans l'exemple précédent, nous avons introduit la ligne 1Ø avant la ligne 2Ø, cependant l'ordre des lignes à l'introduction par le clavier n'a aucune importance, B.E.V.F. se charge toujours de tout remettre en ordre croissant. Pour éditer sur l'écran le listing complet du programme classé par numéros de lignes croissants, tapez la commande suivante:

```
LIST
```

B.E.V.F. affiche:

```
1Ø PRINT 2 + 3  
2Ø PRINT 2 - 3
```

Il est parfois nécessaire de supprimer une ligne dans un programme. Pour cela, il suffit de taper le numéro de la ligne à enlever puis de presser la touche `RETURN`.

Tapez:

```
1Ø  
LIST
```

B.E.V.F. affiche:

```
2Ø PRINT 2 - 3
```

La ligne 1Ø a disparu du programme. Pour réinsérer de nouveau une ligne 1Ø, tapez 1Ø suivi de l'instruction que vous désirez faire exécuter au B.E.V.F. (sans oublier bien sûr la touche `RETURN`).

Entrez l'instruction suivante:

```
1Ø PRINT 2 * 3  
LIST
```

B.E.V.F. affiche:

```
1Ø PRINT 2 * 3  
2Ø PRINT 2 - 3
```

Il n'est pas obligatoire, pour changer la ligne 1Ø, de la supprimer puis de la réécrire avec une nouvelle instruction, il suffit de taper la nouvelle ligne 1Ø (sans oublier `RETURN`), le B.E.V.F. détruira automatiquement l'ancienne instruction.

Entrez l'instruction suivante:

```
1Ø PRINT 3 - 3  
LIST
```

B.E.V.F. affichera:

```
1Ø PRINT 3 - 3  
2Ø PRINT 2 - 3
```

Il est conseillé de laisser un intervalle (par exemple 10) entre chaque numéro de ligne, pour pouvoir, éventuellement, insérer des instructions supplémentaires au programme.

Si vous voulez détruire le programme qui est en mémoire, tapez NEW.

Si vous avez terminé d'exécuter un programme, et que vous êtes sur le point d'en commencer un autre, n'oubliez pas d'insérer la commande NEW. Ceci pour éviter un mélange entre l'ancien et le nouveau programme.

Entrez la commande suivante:

NEW

B.E.V.F. affiche:

]■

Entrez maintenant la commande:

LIST

B.E.V.F. affiche:

]■

Ce qui montre que le programme n'est plus dans la mémoire.

FORMAT DES NOMBRES

Attention, la notation anglaise est utilisée: la virgule française (,) pour séparer la partie entière de la partie décimale est un point (.) en anglais. Abandonnez donc la virgule française et remplacez la par un point.

Nous allons maintenant parler des formats d'affichage et de calcul sur les nombres du B.E.V.F.

La précision des nombres dans un stockage interne est de 9 chiffres. Quand un nombre est affiché, ces 9 chiffres sont affichés; un nombre peut avoir un exposant (puissance de 10).

Avec le B.E.V.F., les nombres réels (appelés aussi nombres à virgule flottante) doivent être compris entre $-1 * 10 \wedge 38$ et $1 * 10 \wedge 38$, sinon un message d'erreur s'affichera. Néanmoins, il est possible qu'en utilisant les additions et les soustractions, vous puissiez générer des nombres aussi grands que $1.7 * 10 \wedge 38$ sans que le message d'erreur se déclenche.

Un nombre dont la valeur absolue est inférieure à $3 * 10 \wedge -39$ est considéré comme égal à zéro par le B.E.V.F.

Enfin, les valeurs des nombres déclarés comme entiers doivent être comprises entre -32767 et 32767.

Quand un nombre s'affiche, le format sur l'écran respecte les règles suivantes.

- 1 - Si le nombre est négatif, un signe (-) est affiché.
- 2 - Si la valeur absolue du nombre est un entier entre 0 et 999 999 999, le nombre est affiché dans un format d'entier.
- 3 - Si la valeur absolue du nombre est plus grande ou égale à 0.01 mais inférieure à 999 999 999.2 le nombre est affiché en format virgule flottante et sans exposant.

4 - Si le nombre ne rentre pas dans les catégories 2 ou 3, la notation scientifique est utilisée. La notation scientifique est utilisée pour l'affichage des nombres réels et se présente comme suit:

SX.XXXXXXXXXESTT

Chaque X est un entier de 0 à 9.

S est le signe du nombre, rien pour un nombre positif et (-) pour un négatif.

Un chiffre non nul est affiché avant la virgule. Les 8 autres chiffres de la mantisse suivent la virgule.

Un E affiché annonce l'exposant, S désigne alors le signe de l'exposant, suivi de deux chiffres (TT), représentant la valeur de l'exposant.

Les zéros inutiles (avant la virgule ou en queue de nombre) ne sont jamais affichés.

Les deux chiffres de l'exposant sont toujours affichés même si le premier est un zéro.

La valeur d'un nombre en notation scientifique est égale à la partie à gauche de E multipliée par 10 élevé à la puissance de l'exposant.

Voici des exemples qui illustrent les règles énoncées précédemment:

<u>NOMBRE</u>	<u>FORMAT D'AFFICHAGE</u>
+1	1
-1	-1
6523	6523
-23.460	-23.46
45.72E5	4572000
1 * 10 ^ 20	1E + 20
-12.34567896 * 10 ^ 10	-1.2345679E + 11
10000000000	1E + 09
9999999999	999999999

Un nombre introduit par le clavier ou utilisé comme une constante dans un programme, peut avoir autant de chiffres désirés, jusqu'à 38 mais seulement les 10 premiers seront significatifs (le 10ième est d'ailleurs arrondi).

Par exemple tapez:

PRINT 1.23456787654321

B.E.V.F. affiche:

1.23456788

EXEMPLE DE GRAPHISME COULEUR

Tapez:

GR

Cela effacera les 20 premières lignes de l'écran et laissera l'affichage des 4 dernières lignes du bas.

Votre ITT 2020 est maintenant en mode graphique couleur large résolution.

Introduisez au clavier:

COLOR = 14

B.E.V.F. affichera le caractère] et le curseur mais B.E.V.F. a sélectionné une couleur

jaune.

Entrez maintenant:

PLOT 20, 20

B.E.V.F. tracera alors un petit rectangle jaune au centre de votre écran. Si le rectangle n'est pas jaune, c'est que votre télévision est mal ajustée, modifiez alors la couleur ou la teinte du récepteur pour obtenir un rectangle de couleur citron.

Tapez ensuite:

HLIN 0, 30 AT 20

B.E.V.F. tracera alors une ligne horizontale entre la colonne 0 et 30 de l'écran et à la ligne 20.

Entrez maintenant:

COLOR = 6

pour changer la couleur, puis tapez:

VLIN 10, 30 AT 30

Nous vous en apprendrons plus sur le graphisme couleur large résolution plus loin dans ce manuel.

Pour revenir au mode texte, tapez:

TEXT

Les caractères apparus sur l'écran proviennent de la transformation des informations graphiques en texte.

Quand vous affichez des réponses à des calculs (PRINT) il est souvent nécessaire d'accompagner ces résultats par du texte explicatif.

Entrez au clavier:

PRINT "UN TIERS EST EGAL A", 1/3

B.E.V.F. répondra:

UN TIERS EST EGAL A .333333333

FORMATS D'AFFICHAGE

Comme nous avons vu ci-dessus, l'insertion d'une virgule (,) dans un ordre PRINT provoque l'affichage de la valeur numérique espacée du texte (champ de tabulation). Si on utilise le point-virgule (;) au lieu de la virgule, le nouvel affichage se sera alors collé au dernier affichage présent sur l'écran.

Essayez les exemples suivants:

PRINT 1, 2, 3

```
1          2          3
```

PRINT 1; 2; 3

```
123
```

PRINT -1; 2; -3

```
-12-3
```


Le petit programme qui suit demande une valeur au clavier et utilise cette valeur dans le calcul et l'affichage de la ligne 2Ø

```
1Ø INPUT R
2Ø PRINT 3.14159 * R * R
RUN
?1Ø
314.159.
```

Expliquons ce qu'il s'est passé; quand B.E.V.F. rencontre l'instruction INPUT, il affiche un point d'interrogation (?) sur l'écran, et attend que vous lui introduisiez un nombre. Quand vous l'avez fait (1Ø a été introduit dans l'exemple) le nombre est assigné à la variable R.

L'instruction suivante est alors effectuée (ligne 2Ø dans l'exemple). Quand la formule qui suit l'ordre PRINT est exécutée la valeur 1Ø est substituée à R à chaque fois que R apparaît dans la formule.

Donc la formule devient:

3.14159 * 1Ø * 1Ø c'est-à-dire 314.159.

Vous aviez déjà deviné, ce programme exemple calcule la surface d'un cercle de rayon R. Si vous voulez calculer la surface de différents cercles, vous pouvez exécuter à nouveau le programme par des ordres RUN successifs.

Mais il y a plus simple, il suffit d'ajouter une ligne au programme:

```
3Ø GOTO 1Ø
RUN
? 1Ø
314.159
? 3
28.27431
? 4.7
69.3977231
?
BREAK IN 1Ø
]
```

En inscrivant l'instruction GOTO 1Ø à la fin du programme vous le faites revenir en ligne 1Ø après l'affichage de chaque calcul de surface.

Nous aurions pu continuer indéfiniment, mais nous avons voulu nous arrêter après le calcul de la troisième surface. L'arrêt est provoqué en tapant CONTROL C au clavier (enfoncer la touche **CTRL** puis sans la relâcher, enfoncer la touche **C**) puis **RETURN**. Cela permet d'interrompre l'exécution d'un programme et de l'arrêter. L'utilisation du CONTROL C permet l'arrêt de n'importe quel programme après l'exécution de l'instruction en cours.

Essayez.

NOMS DES VARIABLES

La lettre R dans le programme que nous venons de faire tourner a été appelée variable. C'est simplement une location mémoire dans l'ordinateur, identifiable par la lettre R. Un nom de variable doit commencer par une lettre de l'alphabet et peut être suivi par

18
tout caractère alphanumérique. Un caractère alphanumérique est une lettre de l'alphabet de A à Z ou un chiffre de 0 à 9.

Un nom de variable peut avoir jusqu'à 238 caractères, mais la reconnaissance du B.E.V.F. se limite aux deux premiers caractères du nom de la variable.

Par exemple, les noms RUE DU CHAT et RUEE se réfèrent à la même variable.

Dans un nom de variable tout caractère alphanumérique après les deux premiers du nom est ignoré, à moins qu'il ne s'agisse d'un "mot réservé". En effet, certains mots du B.E.V.F. sont réservés à des instructions spécifiques. N'utilisez pas ces mots comme noms de variables en partie d'un nom de variable. Par exemple PEND est interdit car END est un "mot réservé".

Les "mots réservés" du B.E.V.F. sont donnés et expliqués dans l'annexe F du manuel. Les noms de variables se terminant par un \$ ou un % ont une fonction spéciale, nous le verrons plus tard dans le chapitre (cf. VARIABLES REELLES, ENTIERES ET ALPHANUMERIQUES).

Voici quelques exemples de noms de variables autorisés et interdits par B.E.V.F.:

AUTORISES

TP
PSTG\$
COUNT
N1%

INTERDITS

TO (un nom de variable ne peut être un mot réservé)
RGOTO (un nom de variable ne peut contenir un mot réservé)

L'assignation des variables peut se faire comme nous l'avons vu avec l'instruction d'assignation LET.

Essayez les exemples suivants:

```
A = 5  
PRINT A, A * 2  
5          10  
LET Z = 7  
PRINT Z, Z-A  
7          2
```

Comme le montrent ces exemples, le LET est une instruction optionnelle pour une assignation.

B.E.V.F. se "mémorise" les valeurs assignées aux variables par ce type d'instructions. La "mémorisation" utilise de la place mémoire pour stocker les valeurs des variables. Cette réservation de place mémoire est conservée jusqu'à ce qu'une des quatre commandes ci-dessous soit effectuée.

- 1 - Une nouvelle ligne de programme est introduite ou bien une ancienne est supprimée.
- 2 - Une commande CLEAR est exécutée
- 3 - Une commande RUN est exécutée
- 4 - Une commande NEW est exécutée

Pour finir, quelque chose d'important: les variables numériques ont automatiquement les valeurs 0 tant qu'elles n'ont pas été assignées.

Essayez ceci:

```
PRINT Q, Q + 2, Q * 2  
0          2          0
```

Une autre instruction: REM, abréviation de REMarque.
Cette instruction est utilisée pour insérer des commentaires, mots ou remarques dans un programme. Quand B.E.V.F. rencontre une instruction REM, le reste de la ligne est ignoré. Cela sert principalement d'aide au programmeur et n'a pas de fonction particulière dans un programme précis.

IF...THEN, LES TESTS

Ecrivons un programme qui vérifie si un nombre introduit par vous au clavier est nul ou non. C'est impossible avec les instructions que nous avons préalablement vues. Nous avons besoin d'une instruction qui exécute un branchement conditionnel sur une autre instruction: c'est l'instruction IF...THEN

Tapez NEW et introduisez le programme suivant:

```
1Ø INPUT B
2Ø IF B = Ø THEN GOTO 5Ø
3Ø PRINT "NON-NUL"
4Ø GOTO 1Ø
5Ø PRINT "NUL"
6Ø GOTO 1Ø
```

A l'exécution de ce programme, un point d'interrogation sera affiché et l'ordinateur attendra que vous lui donniez la valeur affectée à B. Tapez n'importe quelle valeur. L'ITT 2Ø2Ø exécutera alors le test IF...THEN.

Entre le IF et le THEN, il y a une assertion, c'est-à-dire deux expressions séparées par un des symboles suivants:

<u>SYMBOLE</u>	<u>SIGNIFICATION</u>
=	égal à
>	plus grand que
<	plus petit que
<> or >>	différent de
<=	plus petit ou égal à
>=	plus grand ou égal à

L'instruction IF est vraie ou fausse selon que l'assertion est vraie ou fausse. Dans notre présent programme, par exemple, si B vaut zéro alors l'assertion B = Ø est vraie et l'exécution du programme est enchaînée par THEN et l'instruction GOTO 5Ø. Respectant cet ordre l'ordinateur continuera le programme en sautant à la ligne 5Ø et affichera le texte NUL. Puis l'instruction 6Ø GOTO 1Ø renverra le programme à la demande de la ligne 1Ø.

Essayez maintenant le programme suivant pour comparer deux nombres (N'oubliez pas de taper NEW pour détruire l'ancien programme).

```
1Ø INPUT A, B
2Ø IF A <= B THEN GOTO 5Ø
3Ø PRINT "A EST PLUS GRAND"
4Ø GOTO 1Ø
5Ø IF A < B THEN GOTO 8Ø
6Ø PRINT "ILS SONT EGAUX"
7Ø GOTO 1Ø
8Ø PRINT "B EST PLUS GRAND"
9Ø GOTO 1Ø
```

Après l'ordre RUN, la ligne 10 affichera un point d'interrogation et attendra que vous introduisiez deux nombres, séparés par une virgule.

A la ligne 20, si A est plus grand que B, l'assertion $A \leq B$ est fautive, le THEN GOTO 50 est ignoré. Le programme continue donc son exécution à la ligne 30 en affichant le texte A EST PLUS GRAND et finalement, la ligne 40 renvoie l'ordinateur à la ligne 10.

A la ligne 20, si A est égal à B, $A \leq B$ est vrai et l'ordinateur saute à la ligne 50 puis à la ligne 60 car l'assertion $A < B$ est fautive et le texte ILS SONT EGAUX s'affiche. Enfin, la ligne 70 renvoie l'ordinateur en ligne 10 pour recommencer l'exécution du programme.

Si à la ligne 20, A est plus petit que B, $A \leq B$ est vrai et le programme saute la ligne 50. A la ligne 50, $A < B$ est vrai, donc THEN GOTO 80 s'exécute et à la ligne 80, le programme affiche B EST PLUS GRAND et l'ordinateur est renvoyé au début.

Exécutez les deux programmes précédents plusieurs fois. Puis entraînez-vous à en concevoir en utilisant l'instruction IF...THEN.

La manière la plus facile de comprendre comment B.E.V.F. travaille est de créer ses propres programmes!

N'oubliez pas que pour arrêter ces programmes en cours d'exécution, il faut taper la commande CONTROL C puis RETURN.

UN AUTRE EXEMPLE DE COULEUR

Essayons un programme graphique. Notez l'emploi de l'instruction REM pour les commentaires. Les deux points (:) sont utilisés pour séparer plusieurs instructions sur une même ligne. Après avoir introduit le programme ci-dessous, listez le et vérifiez si vous l'avez tapé correctement. Puis exécutez le RUN.

```

100 GR : REM INITIALISE LE MODE GRAPHIQUE
110 HOME : REM EFFACE LE TEXTE
120 X = 0 : Y = 5 : REM INITIALISE LES POSITIONS DE DEPART
130 XV = 2 : REM FIXE LA VITESSE DES X
140 YV = 1 : REM FIXE LA VITESSE DES Y
150 REM CALCUL D'UNE NOUVELLE POSITION
160 NX = X + XV : NY = Y + YV
170 REM SI LA BALLE ATTEINT LES REBORDS ALORS REBOND
180 IF NX > 39 THEN NX = 39 : XV = -XV
190 IF NX < 0 THEN NX = 0 : XV = -XV
200 IF NY > 39 THEN NY = 39 : YV = -YV
210 IF NY < 0 THEN NY = 0 : YV = -YV
220 REM DESSINE EN JAUNE UNE NOUVELLE POSITION
230 COLOR = 14 : PLOT NX, NY
240 REM EFFACE L'ANCIENNE POSITION
250 COLOR = 0 : PLOT X, Y
260 REM CONSERVE LA POSITION ACTUELLE
270 X = NX : Y = NY
280 REM ARRET APRES 250 DEPLACEMENTS
290 I = I + 1 : IF I < 250 THEN GOTO 160
300 PRINT "POUR LISTER VOTRE PROGRAMME, TAPPEZ 'TEXT'"

```

La commande GR connecte votre ITT en graphisme couleur large résolution. Il affiche aussi en noir l'écran de 40 X 40 points, limite l'affichage du texte à une fenêtre de 4 lignes de 40 caractères au bas de l'écran et sélectionne la couleur (COLOR) noire.

HOME est utilisé pour effacer le texte restant sur l'écran et positionner le curseur en haut de la fenêtre d'écran, c'est-à-dire la ligne 20 du mode texte (0 à 19 utilisés pour l'écran graphique).
L'instruction COLOR = des lignes 230 et 250 fixe la couleur des prochains points à dessiner.

L'instruction PLOT NX, NY de la ligne 230 dessine un petit rectangle, de la couleur jaune définie juste avant par l'instruction COLOR =, à la position définie par NX et NY. N'oubliez pas que NX et NY doivent être des nombres compris entre 0 et 39, sinon le rectangle sera hors de l'écran et une erreur se produira.
De même manière, PLOT X, Y en ligne 250 dessine un petit rectangle à la position spécifiée par X et Y. Mais X et Y sont tout simplement les anciennes coordonnées de NX et NY, conservées après le dessin du dernier rectangle, rectangle jaune, mais cette fois-ci en noir à cause de l'instruction COLOR = 0. Le noir étant la couleur du fond de l'écran, le rectangle jaune semble disparaître.

● REMARQUE: pour passer du graphisme couleur en mode texte, tapez:

TEXT

puis RETURN.

La commande TEXT permet le changement du mode graphisme au mode texte. Ne tenez pas compte des divers symboles affichés sur l'écran; il s'agit de la conversion des caractères graphiques en texte.

Si la ligne 290 du programme vous paraît obscure, soyez patient: elle sera expliquée dans les pages qui suivent.

Comme vous avez pu le constater, l'ITT 2020 peut faire plus qu'utiliser des nombres. Nous reviendrons au graphisme couleur après en avoir appris plus sur le B.E.V.F.

FOR...NEXT

Un avantage des ordinateurs réside dans l'aptitude qu'ils ont à exécuter des tâches répétitives. Supposons que nous voulions une table des racines carrées des entiers de 1 à 10. La fonction du B.E.V.F. pour la racine carrée est SQR, s'employant sous la forme SQR (X), où X est un nombre dont on veut calculer la racine carrée.
Nous pourrions écrire le programme qui suit:

```
10 PRINT 1, SQR (1)
20 PRINT 2, SQR (2)
30 PRINT 3, SQR (3)
40 PRINT 4, SQR (4)
50 PRINT 5, SQR (5)
60 PRINT 6, SQR (6)
70 PRINT 7, SQR (7)
80 PRINT 8, SQR (8)
90 PRINT 9, SQR (9)
100 PRINT 10, SQR (10)
```

Le programme effectuera le travail, mais il est cependant particulièrement mal conçu. On peut l'améliorer nettement en utilisant l'instruction IF déjà vue.

Tapez le programme suivant:

```

1Ø N = 1
2Ø PRINT N, SQR (N)
3Ø N = N + 1
4Ø IF N <= 1Ø THEN GOTO 2Ø

```

A l'exécution, l'affichage des résultats est similaire au programme de 1Ø instructions ci-dessus.

Examinons la manière beaucoup plus simple d'obtenir le même résultat.

En ligne 1Ø il y a une instruction LET d'assignation, qui donne à N la valeur 1. A la ligne 2Ø l'ordinateur affiche N et sa racine carrée, en utilisant la dernière valeur affectée à N. La ligne 2Ø est en fait:

```
2Ø PRINT 1, SQR (1)
```

et le résultat est affiché.

Une étrange instruction apparaît à la ligne 3Ø, c'est une assignation spéciale. Mathématiquement $N = N + 1$ n'a pas de sens. Cependant il faut remarquer que dans une instruction LET le signe "=" ne signifie pas égalité, mais "=" veut dire "être remplacé par". L'instruction prend simplement la dernière valeur de N, lui ajoute 1 et l'assigne dans N. Donc, après le premier passage en ligne 3Ø, N devient égal à 2. Comme à la ligne 4Ø N vaut 2, l'assertion $N \leq 1Ø$ est vraie et le programme recule en ligne 2Ø, avec N valant 2.

Le résultat global est que les lignes 2Ø à 4Ø sont répétées, en ajoutant chaque fois 1 à N. Puis quand N prend finalement la valeur 1Ø à la ligne 2Ø alors la ligne d'après incrémente N de 1 et l'assertion de la ligne 4Ø devient fausse, la portion THEN est ignorée et le programme s'arrête.

On appelle cette technique de calcul: ITERATIONS ou BOUCLES. Comme on emploie énormément cette technique en programmation, il y a des instructions spéciales du B.E.V.F. pour les utiliser.

Regardez le programme ci-dessous:

```

1Ø FOR N = 1 TO 1Ø
2Ø PRINT N, SQR (N)
3Ø NEXT N

```

Le résultat d'affichage est exactement le même que celui des deux programmes précédents.

La ligne 1Ø fixe N à 1. La ligne 2Ø fait afficher les valeurs de N et de sa racine carrée. A la ligne 3Ø un nouveau type d'instruction apparaît: NEXT N qui a pour fonction d'incrémenter de 1 la valeur de N, et de remonter, si l'assertion $N \leq 1Ø$ est vraie, à l'instruction FOR. Le nom de N n'est pas spécifique. Toute autre variable peut être utilisée, pour autant que le nom suivant le FOR et le NEXT soit le même. Vous pouvez par exemple, remplacer N par Z1 dans tout le programme, il se serait exécuté exactement de la même manière.

Supposons que vous vouliez maintenant éditer une table des racines carrées pour les entiers pairs entre 1Ø et 2Ø. Le programme suivant s'en chargerait:

```

1Ø N = 1Ø
2Ø PRINT N, SQR (N)
3Ø N = N + 2
4Ø IF N <= 2Ø THEN GOTO 2Ø

```

Notez la similitude entre ce programme et celui qui calculait les racines carrées des 1Ø premiers entiers. Ce programme pourrait aussi s'écrire en utilisant les instructions

d'itération FOR et NEXT.

```
10 FOR N = 10 TO 20 STEP 2
20 PRINT N, SQR (N)
30 NEXT N
```

La différence entre ce programme et celui qui utilise les itérations FOR...NEXT réside dans l'addition du STEP 2. Ceci indique au B.E.V.F. d'additionner 2 à N, à chaque fois, au lieu de 1 comme dans le programme qui précédait.

Si le pas (STEP) n'est pas fixé le B.E.V.F. le considère comme valant 1. Bien sur le pas (STEP) peut être suivi d'une variable.

Maintenant, si vous voulez afficher un compte à rebours commençant à 10, voici un programme qui l'effectuera:

```
10 I = 10
20 PRINT I
30 I = I - 1
40 IF I >= 1 THEN GOTO 20
```

Notez que l'on peut savoir si I est PLUS GRAND ou égal à 1. La raison est simple: la variable I est décrétementée (c'est-à-dire on soustrait) de 1 à chaque fois que la ligne 30 s'exécute. Dans les exemples précédents, on vérifiait si la variable était plus petite ou égale à une certaine valeur plafond.

L'instruction STEP que l'on vient de voir peut aussi être utilisée avec des nombres L'instruction STEP que l'on vient de voir peut aussi être utilisée avec des nombres négatifs. Cela se fait en utilisant un format équivalent aux programmes antérieurs:

```
10 FOR I = 10 TO 1 STEP -1
20 PRINT I
30 NEXT I
```

Les boucles FOR...NEXT peuvent être aussi imbriquées. En voici un exemple:

```
10 FOR I = 1 TO 5
20 FOR J = 1 TO 3
30 PRINT I, J
40 NEXT J
50 NEXT I
```

Remarquez bien que le NEXT J vient avant le NEXT I. C'est dû à l'emboîtement de la boucle J dans la boucle I.

Le programme ci-dessous est erroné. Exécutez-le et vous verrez ce qu'il se passe:

```
10 FOR I = 1 TO 5
20 FOR J = 1 TO 3
30 PRINT I, J
40 NEXT I
50 NEXT J
```

Il ne fonctionne pas car les boucles se croisent au lieu de s'emboîter. En effet lorsque le NEXT I est rencontré toute mémorisation de la valeur de J dans la boucle est perdue.

LES TABLEAUX

24

Il est souvent pratique de pouvoir accéder à un élément d'une table de nombres. B.E.V.F. permet cela à travers l'utilisation des tableaux.

Un tableau est une table de nombres. Le nom de la table, appelé nom du tableau, doit être n'importe quel nom de variable légal: A par exemple; le nom de tableau A est totalement distinct et séparé de la simple variable A, et on peut les utiliser tous deux dans un même programme.

Pour sélectionner un élément de la table, on lie A à un index: c'est-à-dire que pour choisir le Nième élément de la table, on place N entre parenthèses et on le fait suivre A dans l'instruction.

A(N) est le Nième élément du tableau A.

N.B.: dans ce chapitre nous n'indiquons que les tableaux à une dimension, pour les renseignements complémentaires à l'utilisation des tableaux avec B.E.V.F., se référer au chapitre 5: TABLEAUX ET CHAINES DE CARACTERES.

A(N) n'est qu'un élément du tableau A et B.E.V.F. doit être renseigné sur la place à allouer au tableau entier, en fait il demande la taille maximum possible du tableau. C'est ce dont se charge l'instruction DIM.

En utilisant par exemple DIM A(15) on réserve une zone de mouvements pour l'index allant de 0 à 15. Les index de tableaux commencent toujours à 0; le tableau A sera donc dans ce cas-ci, déclaré pour contenir 16 éléments.

Si A(N) est utilisé dans un programme avant d'avoir été DIMensionné alors, B.E.V.F. réserve automatiquement la place pour 11 éléments (index variant de 0 à 10).

Comme exemple d'utilisation des tableaux essayez le programme suivant, qui trie une liste de 8 nombres introduite par vous:

```
90 DIM A(8) : REM FIXE LA DIMENSION DU TABLEAU A 9 ELEMENTS MAXI
100 REM DEMANDE L'INTRODUCTION DE 8 NOMBRES
110 FOR I = 1 TO 8
120 PRINT "ENTREZ UN NOMBRE"
130 INPUT A(I)
140 NEXT I
150 REM COMPARE LES 8 NOMBRES DEUX A DEUX
160 F = 0 : REM INITIALISE L'INDICATEUR D'ORDRE
170 FOR I = 1 TO 7
180 IF A(I) <= A(I + 1) THEN GOTO 240
190 REM INVERSE A(I) ET A(I + 1)
200 T = A(I)
210 A(I) = A(I + 1)
220 A(I + 1) = T
230 F = 1 : REM L'ORDRE N'ETAIT PAS PARFAIT
240 NEXT I
250 REM F = 0 SIGNIFIE L'ORDRE PARFAIT
260 IF F = 1 THEN GOTO 160 : REM RETRIER
270 PRINT : REM SAUTE UNE LIGNE
280 REM AFFICHE LES NOMBRES TRIES
290 FOR I = 1 TO 8
300 PRINT A(I)
310 NEXT I
```

A la ligne 90, B.E.V.F. réserve la place pour 9 valeurs numériques de A(0) à A(8). Les

lignes 110 à 140 introduisent dans l'ordinateur la liste des nombres dans le désordre. Le tri s'effectue entre les lignes 170 et 240, en se déplaçant dans le tableau et en échangeant chaque paire de nombres qui n'est pas en ordre. F est l'indicateur d'ordre parfait: si F vaut 1 le tableau n'est pas trié, et la ligne 260 demande à l'ordinateur d'améliorer le tri. Si une séquence complète entre les lignes 170 et 240 est effectuée sans interchanger deux nombres, le tableau est trié, F est mis à zéro et la liste triée s'affiche sur l'écran.
Notez qu'un indice peut être une variable ou une expression.

" GOSUB...RETURN ", SOUS PROGRAMMES

Encore deux instructions intéressantes: GOSUB et RETURN. Si votre programme utilise plusieurs fois la même séquence d'instructions, vous pouvez utiliser les instructions GOSUB et RETURN pour éviter la répétition de séquences d'instructions identiques dans leur action plusieurs fois dans le programme.

A la rencontre d'une instruction GOSUB, B.E.V.F. saute au numéro de ligne suivant le GOSUB.

Cependant, B.E.V.F. se rappelle où il était dans le programme principal lors de l'appel du sous-programme (GOSUB). Quand l'instruction RETURN s'exécute, B.E.V.F. continue l'exécution en revenant à la première instruction suivant immédiatement le dernier GOSUB exécuté.

Regardez le programme qui suit:

```
20 PRINT "QUEL EST CE PREMIER NOMBRE";
30 GOSUB 100
40 T = N : REM CONSERVE LE NOMBRE INTRODUIT
50 PRINT "QUEL EST LE SECOND NOMBRE"
60 GOSUB 100
70 PRINT "LA SOMME DES DEUX NOMBRES EST" ; T + N
80 STOP : REM FIN DU PROGRAMME PRINCIPAL
100 INPUT N : REM DEBUT DU SOUS PROGRAMME D'INTRODUCTION
110 IF N = INT(N) THEN GOTO 140
120 PRINT "DESOLE LE NOMBRE DOIT ETRE ENTIER".
130 GOTO 100
140 RETURN : REM FIN DU SOUS PROGRAMME
```

Ce programme demande l'introduction par l'utilisateur de 2 entiers et affiche leur somme. Le sous programme est entre les lignes 100 et 140. Le sous programme demande un nombre, si le nombre introduit n'est pas un entier, l'ordinateur continue à le demander jusqu'à satisfaction.

Le programme principal affiche QUEL EST LE PREMIER NOMBRE puis appelle le sous-programme d'introduction du nombre et de vérification d'entrée du nombre entier. La variable N est sauvegardée dans T lors de la sortie du sous-programme, ceci pour éviter la perte du premier nombre lors du 2ème appel du sous-programme. QUEL EST LE SECOND NOMBRE est alors affecté et le sous-programme est encore appelé, cette fois-ci pour introduire le second nombre.

L'instruction suivante du programme est le STOP qui arrête l'exécution du programme en ligne 80. Si l'instruction STOP avait été omise alors, le programme principal serait passé dans son sous-programme (ligne 100) sans qu'il ait été appelé.
Un autre nombre aurait alors été demandé par l'ordinateur et si vous l'aviez introduit

un message d'erreur aurait été affiché car le sous-programme essaierait alors de retourner au programme principal bien qu'il n'ait pas été appelé par un GOSUB.

On peut aussi bien utiliser END que STOP pour séparer un programme principal des sous-programmes. STOP affichera un message indiquant où le programme a été arrêté. END finira le programme sans aucun message. Ces deux instructions redonnent le contrôle de l'ordinateur à l'utilisateur, en affichant le caractère spécifique B.E.V.F. (I) et le curseur.

" READ...DATA...RESTORE ", LE TRAITEMENT DES DONNÉES

Supposons que vous vouliez utiliser dans un programme des constantes qui ne changent jamais lors de l'exécution, mais qui sont éventuellement facilement modifiables. B.E.V.F. contient des instructions spéciales pour réaliser ce traitement des données: les instructions READ et DATA.

Etudiez le programme suivant:

```

10 PRINT "CHOISISSEZ UN NOMBRE"
20 INPUT G
30 READ D
40 IF D = -999999 THEN GOTO 90
50 IF D <> G THEN GOTO 30
60 PRINT "BRAVO"
70 END
90 PRINT "MAUVAIS, ESSAYEZ ENCORE".
95 RESTORE
100 GOTO 10
110 DATA 1,393, -39, 28, 391, -8, 0, 3.14, 90
120 DATA 89, 5, 10, 15, -34, -999999

```

Voilà ce qui se passe à l'exécution du programme: quand l'instruction READ est rencontrée, l'effet est similaire à l'instruction INPUT, mais au lieu d'attendre un nombre introduit par vous au clavier, l'ordinateur va le chercher dans la table précédé par les instructions DATA.

La première fois que la lecture de la table (READ) est demandée, l'ordinateur va chercher le nombre qui suit la première instruction DATA. Au second passage sur READ, c'est le second nombre suivant la première instruction READ qui est lu. Quand les données de la première instruction DATA ont été lues entièrement alors, la lecture se fera dans les données de la seconde table DATA. Les données (DATA) sont toujours lues séquentiellement et il peut y avoir autant de DATA que vous le désirez dans un programme.

L'objet de ce programme est un petit jeu où vous essayez de deviner un nombre contenu dans les instructions DATA. Pour chaque essai, l'ordinateur décrit sa table de données jusqu'à ce qu'il trouve le nombre de la table égal au nombre introduit par vous. Si l'ordinateur lit -999999 c'est que tous les nombres de la table ont été lus et qu'un nouvel essai doit être effectué. Avant de retourner à la ligne 10 pour un autre essai, il faut indiquer à l'ordinateur que la lecture de la table doit s'effectuer à nouveau à partir du premier élément suivant le premier DATA. C'est la fonction de l'instruction RESTORE. Après qu'un RESTORE ait été exécuté, la première donnée lue sera le premier nombre suivant la première instruction DATA.

Les instructions DATA peuvent être placées n'importe où dans le programme. Seule l'instruction READ peut utiliser les DATA, et sinon DATA est toujours ignoré dans l'exécution d'un programme.

VARIABLES RÉELLES, ENTIÈRES ET ALPHANUMÉRIQUES

B.E.V.F. utilise trois types de variables. Jusqu'à maintenant nous n'en avons vu qu'un: le type réel.

Les nombres réels sont affichés avec au plus 9 décimales de précision et peuvent atteindre des quantités jusqu'à 10^38 à la puissance 38.

B.E.V.F. convertit un nombre décimal en binaire pour ses besoins puis de binaire en décimal pour vous l'afficher. A cause des erreurs d'arrondis, les séquences de calcul de racine carrée, division et puissance peuvent parfois ne pas donner la valeur exacte du nombre que vous attendez.

Le nombre de chiffres après la virgule peut être fixé en arrondissant la valeur avant de l'afficher (PRINT).

La formule fixant la virgule est:

$$X = \text{INT} (X * 10^D + .5) / \text{INT} (10^D + .5)$$

(Rappelez-vous que pour le système anglo-saxon le point (.) équivaut à la virgule (,) décimale des nombres que vous utilisez.)

Dans la formule ci-dessus, D est le nombre de chiffres désirés après la virgule.

Voici un moyen plus rapide pour exécuter cette formule:

$$P = 10^D \\ X = \text{INT} (X * P + .5) / P$$

Avec $P = 10$ on a un chiffre, $P = 100$ 2 chiffres, $P = 1000$ 3 chiffres après la virgule.

La formule est valable pour $X \geq 1$ et $X < 999\,999\,999$

Un programme limitant le nombre de chiffres après la virgule est donné au prochain paragraphe de ce chapitre.

La table ci-dessous résume les trois types de variables traitées par B.E.V.F.

<u>TYPE</u>	<u>SYMBOLE D'APPARTENANCE AU TYPE</u>	<u>EXEMPLE</u>
alphanumériques (0 à 255 caractères)	\$	A\$ ALPHA\$
entières (entre -32767 et +32767)	%	B% C1%
réelles 9 chiffres, exposants de -38 à +38)	au choix!...	C BOY

Une variable entière ou alphanumérique doit être respectivement suivie du signe % ou \$, à chaque appel de cette variable. Par exemple X, X% et X\$ sont des variables différentes. Les variables entières sont interdites avec les instructions FOR et DEF. Le grand intérêt des variables entières est le gain de place mémoire qu'elles apportent lors de la création d'importants tableaux.

Toute opération arithmétique s'effectue en réels, les entiers et les variables entières sont d'abord convertis en réels avant d'être utilisés dans un calcul. Les fonctions SIN, COS, ATN, TAN, SQR, LOG, EXP et RND convertissent aussi les arguments en réel pour calculer et afficher leurs résultats. Quand un nombre est converti en entier il est arrondi par défaut.

Par exemple:

```
I% = -.999          A% = -.01
PRINT I%           PRINT A%
Ø                  -1
```

Assigner un nombre réel dans une variable entière et faire afficher cette variable (PRINT) équivaut à appliquer la fonction partie entière à ce nombre réel. Une conversion automatique (=) entre chaînes de caractères et variables numériques est impossible et un message d'erreur s'affichera. Cependant il existe des fonctions spécifiques qui effectuent ce type de conversions.

CHAÎNES DE CARACTÈRES

Une suite de caractères est considérée comme un littéral. Une chaîne de caractères est un littéral entre guillemets. Exemple de chaînes de caractères:

```
"BONJOUR", "ITT 2Ø2Ø", "C'EST UN ESSAI"
```

Comme les variables numériques; les variables de chaînes de caractères (alphanumériques) peuvent s'assigner. On les distingue des variables numériques par le caractère \$ qui termine leur nom.

Essayez:

```
A$ = "BONJOUR"
PRINT A$
BONJOUR
```

On a dans cet exemple assigne à A\$ la chaîne BONJOUR. Maintenant qu'A\$ contient une chaîne, on peut en extraire sa longueur (le nombre de caractères contenu dans la chaîne), en faisant comme suit:

```
PRINT LEN (A$), LEN ("OUI")
      7          3
```

La fonction LEN affiche un entier indiquant le nombre de caractères contenus dans la chaîne. Le nombre de caractères d'une chaîne doit être compris entre Ø et 255. Une chaîne de Ø caractères est une chaîne nulle. Avant qu'une chaîne de caractères soit utilisée dans un programme, elle est considérée comme nulle par l'ordinateur. Faire afficher une chaîne de caractères nulle n'aura pas d'effet sur l'écran et le curseur ne sera pas avancé.

Essayez:

```
PRINT LEN (Q$) ; Q$ ; 3
```

Ø3

On peut aussi créer une chaîne de caractères nulle avec l'instruction:

```
Q$ = "" au lieu de LET Q$ = ""
```

Annuler une chaîne de caractères peut s'utiliser pour libérer la place mémoire prise par une chaîne de caractères non nulle. Mais vous risquez certains ennuis en assignant une chaîne nulle dans une variable alphanumérique (voir au chapitre 7 avec l'instruction IF).

Il est souvent utile d'extraire une partie d'une chaîne alphanumérique pour la manipuler. Maintenant qu'A\$ contient BONJOUR faisons afficher les 3 premiers caractères de A\$.

Tapez:

```
PRINT LEFT$(A$, 3)
BON
```

LEFT\$(A\$, N) est une fonction qui extrait une sous-chaîne composée de N caractères les plus à gauche de la variable alphanumérique (A\$ dans ce cas).

Voici un autre exemple:

```
FOR N = 1 TO LEN(A$) : PRINT LEFT$(A$,N) : NEXT N
```

```
B
BO
BON
BONJ
BONJO
BONJOU
BONJOUR
```

Comme A\$ a 7 caractères, la boucle tournera avec N = 1, 2, 3 7. Au premier passage le premier caractère de la chaîne seul s'affichera. Au deuxième passage, les deux premiers, etc.

Une autre fonction sur chaînes de caractères: RIGHT\$.

RIGHT\$(A\$, N) affiche les N caractères les plus à droite de la chaîne A\$. Remplacez dans l'exemple ci-dessus LEFT\$ par RIGHT\$ et regardez le résultat.

Une instruction permet aussi d'extraire des caractères du milieu de la chaîne alphanumérique: MID\$.

Essayez:

```
FOR N = 1 TO LEN(A$) : PRINT MID$(A$, N) : NEXT N
```

MID\$(A\$, N) extrait une sous-chaîne commençant au Nième caractère de A\$ et se terminant au dernier caractère de A\$. La première position d'une chaîne est 1 et la dernière 255. Il est souvent utile de pouvoir extraire le Nième caractère d'une chaîne. C'est possible en utilisant la fonction MID\$ avec 3 arguments:

```
MID$(A$, N, 1)
```

Le troisième argument désignant le nombre de caractères qui doivent être extraits, à partir du caractère de rang N.

Exemple:

```

FOR I = 1 TO LEN (A%) : PRINT MID% (A%, N, 1), MID% (A%, N, 2) : NEXT N
B      BO
O      ON
N      NJ
J      JO
O      OU
U      UR
R      R

```

Référez-vous au chapitre 5 pour en savoir plus sur LEFT%, MID%, RIGHT%.

La concaténation des chaînes de caractères est aussi possible grâce à l'opérateur (+).

Introduisez au clavier:

```

B% = A% + " " + "JEAN"
PRINT B%
BONJOUR JEAN

```

La concaténation est spécialement utile si vous désirez extraire des "morceaux" de chaînes de caractères et les réunir à nouveau avec les ajouts.

Par exemple:

```

C% = RIGHT% (B%, 4) + "-" + LEFT% (B%, 7)
PRINT C%
JEAN - BONJOUR

```

Il est parfois indispensable de pouvoir convertir un nombre en sa représentation alphanumérique et vice-versa.

Les fonctions VAL et STR% font ces conversions.

Essayez:

```

STRING% = "567.8"
PRINT VAL (STRING%)
567.8

```

```

STRING% = STR% (3.1415)
PRINT STRING% , LEFT% (STRING% , 5)
3.1415      3.141

```

La fonction STR% peut être utilisée pour changer le format d'affichage et d'entrée. Vous pouvez convertir un nombre en une chaîne de caractères puis utiliser les fonctions LEFT%, RIGHT%, MID% et de concaténation pour formater le nombre comme désiré.

Le programme ci-dessous concrétise l'emploi des chaînes de caractères pour formater l'affichage numérique:

```

100 INPUT "ENTREZ UN NOMBRE:" ; X
110 PRINT : REM SAUTE UNE LIGNE
120 PRINT "COMBIEN DE CHIFFRES APRES LA"
130 INPUT "VIRGULE VOULEZ VOUS VOIR" ; D
140 GOSUB 1000
150 PRINT " " : REM SEPARATEUR
160 GOTO 100
1000 X% = STR% (X) : REM CONVERTIT LE NOMBRE EN ALPHANUMERIQUE
1010 REM TROUVE LA POSITION DE L'EXPOSANT E, S'IL EXISTE

```

```

1020 FOR I = 1 TO LEN (X%)
1030 IF MID% (X%, I, 1) <> "E" THEN NEXT I
1040 REM ON A TROUVE L'EXPOSANT
1050 REM TROUVE LA POSITION DE LA VIRGULE , SI ELLE EXISTE
1060 FOR J = 1 TO I-1
1070 IF MID% (X%, I, 1) <> "." THEN NEXT J
1080 REM A TROUVE LA VIRGULE
1090 REM EST CE QU'IL Y A DES CHIFFRES A DROITE
1100 IF J + D = I - 1 THEN N = J + D : GOTO 1130 : REM OUI
1110 N = I - 1 : REM NON, ALORS AFFICHER TOUS LES CHIFFRES
1120 REM AFFICHE LE NOMBRE ET SON EXPOSANT
1130 PRINT LEFT% (X%, N) + MID% (X%, I)
1140 RETURN

```

Le programme ci-dessus utilise le sous-programme 1000 pour afficher une variable réelle X tronquée (non arrondie) au Dième chiffre après la virgule. Les variables X%, I et J sont des variables locales dans le sous-programme.

La ligne 1000 convertit la variable réelle X en variable alphanumérique X%. Les lignes 1020 et 1030 sondent la chaîne pour savoir si E est présent. I pointe sur E, ou sur LEN (X%) + 1 si E n'existe pas. Les lignes 1060 et 1070 sondent la chaîne pour savoir si la virgule est présente. J est positionné sur la virgule ou sur I - 1 s'il n'y a pas de virgule.

La ligne 1100 vérifie s'il existe au moins D chiffres après la virgule. S'ils existent, la chaîne alphanumérique représentant le nombre doit être tronquée à la position J + D, c'est-à-dire D positions à droite de J qui indique la virgule. La variable N est mise à J + D.

S'il y a moins de D chiffres à droite de la virgule, c'est le nombre en entier qui est affiché. La ligne 1110 fixe la variable N à cette longueur (I - 1). Finalement la ligne 1130 affiche la variable X comme concaténation de deux sous-chaînes. LEFT% (X%, N) affiche les chiffres significatifs du nombre et MID% (X%, I) affiche la partie de l'exposant s'il existe.

STR% peut aussi servir à savoir combien un nombre prendra de place à l'affichage. Par exemple:

```

PRINT LEN (STR% (33333 . 157))
9

```

Si dans un programme un utilisateur entre une question telle que:

QUEL EST LE VOLUME D'UN CYLINDRE DE RAYON 5.36 METRES ET DE HAUTEUR 5.1 METRES?

Vous pouvez utiliser la fonction VAL pour extraire les valeurs numériques des sous-chaînes 5.36 et 5.1 utilisées dans la question ci-dessus. (Pour plus de renseignements, voyez le chapitre 5.)

Le programme suivant trie une liste de chaînes de caractères et l'affiche dans l'ordre alphabétique. Ce programme ressemble énormément à celui que vous avez employé pour trier des valeurs numériques.

```

100 DIM A% (16)
110 FOR I = 1 TO 15 : READ A% (I) : NEXT I
120 F = 0 : I = 1
130 IF A% (I) <= A% (I + 1) THEN GOTO 180
140 T% = A% (I + 1)
150 A% (I + 1) = A% (I)

```

```

160 A$(I) = T$
170 F = 1
180 I = I + 1 : IF I <= 15 THEN GOTO 130
190 IF F = 1 THEN GOTO 120
200 FOR I = 1 TO 15 : PRINT A$(I) : NEXT I
220 DATA ITT 2020, CHAT, CHIEN, ALIAS, ORDINATEUR
230 DATA BASIC, BASIC ETENDU, LUNDI, ENNEMI, " ***REPOSE** "
240 DATA COMPUTER, AIDE, BATAILLE, PROGRAMME, FIN

```

ENCORE PLUS DE GRAPHISME COULEUR

Dans deux exemples, nous vous avons expliqué comment l'ITT 2020 peut traiter le graphisme couleur aussi bien que le texte. En mode Graphique, l'ITT 2020 peut affecter jusqu'à 1600 petits rectangles, dans 16 couleurs possibles, sur une grille de 40 X 40. Il vous laisse aussi la possibilité d'utiliser les 4 lignes de texte restantes sur l'écran. L'axe des abscisses (x) est orienté normalement, 0 le plus à gauche, 39 le plus à droite. L'axe des ordonnées (y) est INVERSE: 0 est le plus en HAUT et 39 le plus en BAS.

Essayez ce programme:

```

10 GR : REM SELECTION DU MODE GRAPHISME COULEUR; AFFICHE L'ECRAN DE 40 X 40 EN NOIR,
    LAISSE 4 LIGNES DE TEXTE
20 HOME : REM EFFACE LE TEXTE DU BAS DE L'ECRAN
30 COLOR = 1 : PLOT 0, 0 : REM RECTANGLE BLEU OUTREMER EN X = 0, Y = 0
40 LIST 30 : GOSUB 1000
50 COLOR = 2 : PLOT 39, 0 : REM RECTANGLE VERT BOUTEILLE EN X = 39, Y = 0
60 HOME : LIST 50 : GOSUB 1000
70 COLOR = 12 : PLOT 0, 39 : REM RECTANGLE ROUGE CLAIR EN X = 0, Y = 39
80 HOME : LIST 70 : GOSUB 1000
90 COLOR = 9 : PLOT 39, 39 : REM RECTANGLE BLEU CLAIR EN X = 39, Y = 39
100 HOME : LIST 90 : GOSUB 1000
110 COLOR = 13 : PLOT 19, 19 : REM RECTANGLE VIEUX ROSE AU CENTRE
120 HOME : LIST 110 : GOSUB 1000
130 HOME : PRINT "DESSINEZ VOS PROPRES RECTANGLES"
140 PRINT "N'oubliez pas que X et Y doivent être >= 0 et <= 39"
150 INPUT "ENTREZ X, Y" ; X, Y
160 COLOR = 8 : PLOT X, Y : REM RECTANGLES MARRON
170 PRINT "TAPEZ 'CTRLC' ET RETURN POUR ARRÊTER"
180 GOTO 150
1000 PRINT "***TAPEZ UNE TOUCHE POUR CONTINUER***" ; : GET A$ : RETURN

```

Après avoir introduit ce programme faites le lister pour vérifier qu'il n'y a pas d'erreurs. Vous pouvez le conserver sur cassette (SAVE) pour des besoins futurs.

Exécutez le programme.

La commande GR sélectionne le mode graphique couleur large résolution.

La commande COLOR définit la couleur du prochain point à dessiner. Cette couleur reste jusqu'à ce qu'une autre commande COLOR soit utilisée. Par exemple la couleur de la ligne 160 reste toujours la même indépendamment du nombre de rectangles à dessiner. La valeur de l'expression suivant COLOR doit être comprise entre 0 et 255 sinon un message d'erreur s'affichera. Il n'y a cependant que 16 couleurs numérotées de 0 à 15.

Changez le programme en ré-introduisant les lignes 150 et 160 comme suit:

```

150 INPUT "ENTREZ X, Y, COULEUR :" ; X, Y, Z

```


160 COLOR = Z : PLOT X, Y

A l'exécution du programme vous serez en mesure de choisir la couleur du rectangle à dessiner. Nous verrons plus loin la gamme de couleurs de l'ITT 2020.

L'instruction PLOT X, Y dessine un petit rectangle de la couleur définie par la dernière instruction contenant COLOR, et à la position spécifiée par les expressions X et Y. N'oubliez pas que X et Y doivent être des nombres compris entre 0 et 39.

L'instruction GET de la ligne 1000 est similaire à l'instruction INPUT. Mais elle n'attend qu'un SEUL caractère d'une touche du clavier, quand vous le lui donnez, elle l'assigne à la variable suivant le GET. Et l'exécution continue.

Il NE FAUT PAS appuyer sur **RETURN**.

GET est utilisé en ligne 1000 uniquement pour arrêter le programme en attendant qu'une touche soit enfoncée.

N'oubliez pas que pour revenir au mode texte il faut taper TEXT puis RETURN. Le signe spécifique du B.E.V.F. (j) réapparaîtra.

Entrez le programme suivant et exécutez le (RUN) pour afficher sur l'écran la gamme des couleurs disponibles de l'ITT 2020.

```
10 GR : HOME : REM INITIALISE LE MODE GRAPHISME
20 FOR I = 0 TO 31
30 COLOR = I/2
40 VLIN 0, 39 AT I
50 NEXT I
60 FOR I = 0 TO 14 STEP 2 : PRINT TAB (I*2 + 1) ; I ; : NEXT I
70 PRINT
80 FOR I = 1 TO 15 STEP 2 : PRINT TAB (I*2 + 1) ; I ; : NEXT I
90 PRINT : PRINT "COULEURS STANDARD ITT 2020"
```

Les colonnes colorées sont déterminées au double de leur largeur normale. La colonne la plus à gauche est noire, la colonne la plus à droite est blanche. Et en fonction de votre réglage TV, la deuxième colonne (COLOR = 1) sera bleu outremer et la troisième (COLOR = 2) sera vert bouteille. Réglez votre TV pour obtenir ces couleurs.

Dans le programme ci-dessus, nous utilisons une instruction de la forme VLIN Y1, Y2 AT X à la ligne 40. Cette instruction trace une ligne verticale de l'ordonnée Y1 à l'ordonnée Y2 à la position horizontale X.

Y1, Y2, X doivent varier entre 0 et 39.

Y2 peut être plus grande, égale ou inférieure à X1. La commande HLIN W1, X2 AT Y est similaire à VLIN dans sa forme mais trace une ligne horizontale.

Notez que l'ITT 2020 trace une ligne aussi facilement qu'il dessine un petit rectangle.

GRAPHISME COULEUR HAUTE RÉOLUTION

Maintenant que vous êtes familiarisé avec le graphisme couleur large résolution de votre ITT 2020, vous comprendrez facilement le graphisme couleur haute résolution.

Les instructions sont très semblables à celles que vous connaissez déjà à la différence d'un H (pour haute résolution) qu'il faut ajouter devant chaque instruction.

Par exemple, la commande HGR sélectionne le mode graphique couleur haute résolution, nettoie l'écran haute résolution et laisse quatre lignes de texte en bas de l'écran.

Dans ce mode vous pouvez dessiner les points sur une grille de 360 points d'abscisse et de 160 points d'ordonnées.

Cela vous permet un dessin beaucoup plus précis que sur la grille 40 X 40 du mode graphi-³⁴
que couleur large résolution.

La commande TEXT retourne au mode texte.

En plus de l'écran HGR, vous disposez aussi d'un second écran de haute résolution que
vous pouvez utiliser si votre appareil contient au moins 24K octets de mémoire.

Le mode graphisme haute résolution pour cette "seconde page" de mémoire se sélectionne par
la commande: HGR2, qui nettoie l'écran et vous donne une surface de dessin de 360 sur
l'abscisse et 192 sur l'ordonnée, mais sans affichage possible de texte.

Tapez TEXT pour revenir au mode texte.

Ces extraordinaires performances graphiques nous ont obligés à sacrifier quelques couleurs.
Les couleurs sont sélectionnées par une instruction du type:

```
HCOLOR = N
```

Où N est un nombre entier entre 0 (noir) et 7 (blanc).

Référez-vous au chapitre 8 pour la liste complète des couleurs disponibles. A cause des
différences de construction des TV couleurs, les couleurs annoncées peuvent varier d'un
poste à l'autre. Il y a pour dessiner les points en haute résolution une instruction
simple. Exécutez la commande suivante pour la voir:

```
HCOLOR = 3  
HGR  
HPLOT 130, 100
```

La dernière commande a dessiné un point haute résolution de couleur blanche au point
X = 130 et Y = 100.

Comme en large résolution l'axe X = 0 est le bord gauche de l'écran et est orienté vers
la droite. L'axe Y = 0 est le bord haut de l'écran, étant orienté vers le bas.

La valeur maximale des X est 359 et celle des Y est 191 (le maximum des Y est de 159 en
haute résolution + 4 lignes de texte).

Tapez maintenant:

```
HPLOT 20, 15 TO 145, 80
```

Une ligne droite se trace en blanc du point X = 20, Y = 15. Au point X = 145, Y = 80.

HPLOT peut tracer des lignes entre deux points quelles que soient leurs positions sur
l'écran (lignes horizontales, verticales, obliques).

Si vous voulez tracer une autre ligne partant de l'extrémité de la droite que vous venez
de dessiner, tapez:

```
HPLOT TO 12, 80
```

Cette instruction trace une ligne droite du DERNIER point dessiné sur l'écran au point
12, 80, elle reprend aussi la couleur du dernier point dessiné.

Vous pouvez aussi chaîner ces commandes dans une instruction.

Essayez:

```
HPL0T 0,0 TO 359,0 : HPL0T TO 359,159 : HPL0T TO 0,159 : HPL0T TO 0,0
```

Vous verrez l'écran entouré d'une bordure blanche.

Voici un programme qui dessine un motif moiré sur votre écran:

```
80 HOME : REM EFFACE LE TEXTE SUR L'ECRAN  
100 VTAB 24 : REM POSITIONNE LE CURSEUR EN BAS DE L'ECRAN
```

```

120 HGR : REM SELECTIONNE LE MODE GRAPHIQUE HAUTE RESOLUTION
140 A = RND (1) * 359 : REM CHOISIT UN X POUR "CENTRE"
160 B = RND (1) * 159 : REM CHOISIT UN Y POUR "CENTRE"
180 I% = (RND (1) * 4) + 2 : REM CHOISIT UN PAS
200 HTAB 15 : PRINT "PAS DE" ; I% ;
220 FOR X = 0 TO 358 STEP I% : REM INCREMENT DES X
240 FOR S = 0 TO 1 : REM 2 LIGNES, DE X ET X + 1
260 HCOLOR = 3 * S : REM PREMIERE LIGNE NOIRE? SECONDE BLANCHE
280 REM TRACE UNE LIGNE DU "CENTRE" AU BORD OPPOSE
300 HPLOT X + S, 0 TO A,B : HPLLOT TO 359 - X - S, 159
320 NEXT S, X
340 FOR Y = 0 TO 158 STEP I% : REM INCREMENT DES Y
360 FOR S = 0 TO 1 : REM 2 LIGNES, DE Y ET Y + 1
380 HCOLOR = 3 * S
400 REM TRACE UNE LIGNE DU "CENTRE" AU BORD OPPOSE
420 HPLOT 359, Y + S TO A,B : HPLLOT TO 0, 159 - Y - S
440 NEXT S, Y
460 FOR PAUSE = 1 TO 1500 : NEXT PAUSE : REM BOUCLE D'ATTENTE
480 GOTO 120 : REM VA DESSINER UN AUTRE MOTIF

```

Ce programme est assez long, faites bien attention en le tapant et pour le vérifier faites le lister par portions (LIST 0, 320 par exemple).

Après vérification exécutez le (RUN)

VTAB et HTAB sont des commandes de déplacement du curseur, que l'on utilise pour afficher les caractères à une position sur l'écran. VTAB 1 place le curseur à la première ligne et HTAB 40 à la dernière colonne (la plus à droite sur l'écran).

Dans une instruction PRINT comme à la ligne 200, vous avez besoin d'un point-virgule pour éviter un saut de ligne qui déplacerait les messages.

La fonction RND (N) où N est un nombre quelconque et positif, donne un nombre aléatoire entre 0 et .999999999 (Référez-vous au chapitre 10 pour en savoir plus sur RND).

La ligne 180 assigne à la variable entière I% un nombre aléatoire entre 2 et 5 (un nombre est toujours arrondi par défaut quand il est converti en entier). Le pas d'une boucle FOR...NEXT n'est pas forcément un entier mais les résultats sont plus prévisibles en utilisant un nombre entier.

Comme vous le constatez en lignes 320 et 440, il suffit d'une instruction NEXT pour plusieurs instructions FOR. Vérifiez que vous avez, dans l'instruction NEXT, indiqué les variables dans le bon ordre pour éviter les erreurs de boucles croisées.

La ligne 460 est une boucle d'attente pour vous laisser le temps d'admirer le dessin avant que l'ordinateur en commence un autre. A chaque fois que l'ordinateur passe sur la ligne 480 il est renvoyé à la ligne 120 où HGR nettoie l'écran pour préparer le dessin d'un autre motif.

Pour arrêter le programme tapez CTRL puis TEXT
C

Imaginez les changements de ce programme!

Par exemple, après l'avoir conservé (SAVE) sur une cassette ou sur le disque, essayez de faire varier aléatoirement la valeur de la couleur (HCOLOR).

Bonne programmation!

Notes



CHAPITRE 2

DEFINITIONS

DÉFINITIONS SYNTAXIQUES ET ABRÉVIATIONS (Pour une liste alphabétique voir l'annexe N)

Les définitions suivantes utilisent des métasymboles tels que `{}` ou `\` pour indiquer sans aucune ambiguïté les structures et rapports existants dans le B.E.V.F.

Un symbole spécial `:=` indique le début d'une définition complète ou incomplète du terme à gauche du symbole `:=`

<code> </code>	<code>:=</code>	métasymbole de séparation indiquant l'alternance des éléments (peut représenter le OU du langage parlé).
<code>[]</code>	<code>:=</code>	métasymboles entourant un élément facultatif.
<code>{ }</code>	<code>:=</code>	métasymboles entourant un élément qui peut être répété.
<code>\</code>	<code>:=</code>	métasymbole entourant un élément dont on utilise la <u>VALEUR</u> (la valeur de X s'écrit alors <code>\ X \</code>).
<code>~</code>	<code>:=</code>	métasymbole indiquant la nécessité d'un blanc (espace).
métasymbole	<code>:=</code>	<code> [] { } \ ~</code>
lettre minuscule	<code>:=</code>	<code>a b c d e f g h i j k l m n o p q r s t u v w x y z</code>
métasymbole	<code>:=</code>	lettre minuscule
chiffre	<code>:=</code>	<code>1 2 3 4 5 6 7 8 9 ø</code>
métanom	<code>:=</code>	<code>{ métasymbole } [chiffre]</code>
métasymbole	<code>:=</code>	un chiffre chaîné à un métanom
symboles spéciaux utilisés par le B.E.V.F.	<code>:=</code>	spéciaux
spéciaux	<code>:=</code>	<code>! # \$ % & ' () * : = - _ + ; ? / > . < , ^ '"</code> Les caractères de contrôle (caractères devant être tapés en tenant enfoncée la touche CTRL) et les caractères sans effet sont aussi des SPECIAUX. B.E.V.F. utilise le crochet droit (]) simplement comme caractère de reconnaissance du langage, il est utilisé dans ces pages comme un métasymbole.
lettre	<code>:=</code>	<code>A B C D E F G H I J K L M N O P Q R S T U V W X Y Z</code>
caractère	<code>:=</code>	lettre chiffre spéciaux
caractère alphanumérique	<code>:=</code>	lettre chiffre
nom	<code>:=</code>	lettre [{ lettre chiffre }]

Un nom peut avoir jusqu'à 238 caractères de longueur. Pour distinguer deux noms, le B.E.V.F. ne regarde que les deux premiers caractères des noms. B.E.V.F. ne voit pas de différences entre BONJOUR et BONTE. Cependant même la partie ignorée d'un nom ne doit pas contenir le caractère " (guillemet) ou un mot réservé du B.E.V.F. (voir la liste à l'annexe A).

entier

:= [+/-] {chiffre}

Les entiers doivent varier entre -32767 et 32767. Lors de la conversion réel à entier le B.E.V.F. applique en fait la fonction partie entière (INT), le nombre réel est alors arrondi par défaut à sa valeur entière. Néanmoins, ce n'est pas vrai pour des nombres approchant de vraiment très près la valeur de l'entier supérieur.

Par exemple:

```
A% = 123.999 999 959 999
PRINT A%
123
```

```
A% = 123.999 999 96
PRINT A%
124
```

Un entier utilise 2 octets (16 bits) en mémoire centrale.

un nom de variable entière

:= name %

Un réel peut être stocké dans une variable entière, mais B.E.V.F. convertit d'abord ce réel en entier.

réel

:= [+ -] {chiffre}{[.]chiffre} [E[+ -] chiffre [chiffre]]

:= [+ -] [{chiffre}] . [{chiffre}] [E[+ -] chiffre [chiffre]]

La lettre E des nombres réels indique la présence d'un exposant (E est une abréviation de * 10^Λ)

Le nombre suivant E est la puissance. Avec B.E.V.F. les réels doivent être compris entre -1E38 et 1E38, sinon le message ?OVERFLOW ERROR de dépassement de capacité s'affichera. En utilisant les additions et les soustractions il est possible de générer des nombres aussi grands que 1.7E38 sans que le message d'erreur s'affiche.

Un réel dont la valeur absolue est plus petite que 2.9388E - 39 vaudra Ø pour le B.E.V.F. B.E.V.F. reconnaît les caractères suivants comme

des réels nuls:

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

L'élément d'un tableau M(.) est le même que M(∅)
En complément de la liste des formats ci-dessus,
les formats suivants sont équivalents à ∅ lors
d'une réponse à INPUT au lieu d'une lecture de
DATA:

+ - E +E -E espace E+ E- +E+ +E- -E+
-E-

L'instruction GET considère comme valant ∅ les
caractères suivants:

. + - E

A l'affichage d'un nombre 9 chiffres maximum
seront sur l'écran (exposant exclu).
Le dernier chiffre est arrondi. Les zéros à
gauche du premier chiffre significatif après la
virgule ne seront pas représentés.
S'il n'y a pas de chiffres significatifs à droi-
te de la virgule, la virgule (.) n'est pas affi-
chée.



L'arrondi peut être quelquefois curieux:

```
PRINT 99 999 999.9
99 999 999.9
PRINT 99 999 999.9∅
1∅∅ ∅∅∅ ∅∅∅
PRINT 11.111 111 45∅ ∅∅
11.111 111 5
PRINT 11.111 111 451 9
11.111 111 4
```

(les espaces dans les nombres sont ajoutés pour
la clarté).

Si une valeur absolue d'un nombre réel compris
entre ∅1 et 999 999 999.2 l'affichage se fait
en virgule flottante, c'est-à-dire qu'aucun
exposant n'apparaît.

Entre .∅ 1∅∅ ∅∅∅ ∅∅∅ 5 et .∅ 999 999 999 les
réels sont affichés avec 1∅ chiffres.

C'est la seule exception à la règle des 9 chif-
fres affichés. (Exposant exclu).

Si vous essayez d'utiliser un nombre de plus de
38 chiffres tel que 211.11111111111111111111111111111111
111111111111

le message ?OVERFLOW ERROR s'affichera (dépasse-
ment de capacité) bien que le nombre soit compris
entre -1E38 et 1E38

Le message se reproduira même si les chiffres
à droite de la virgule sont des ∅, comme:

```
211.∅∅∅∅∅∅∅∅∅∅∅∅∅∅∅∅∅∅∅∅∅∅∅∅∅∅∅∅∅∅∅∅∅∅∅∅∅∅∅∅∅∅∅∅∅∅∅∅∅∅∅
```

	Si le premier chiffre est un 1 et le second inférieur ou égal à 6, les nombres de 39 chiffres sont acceptés sans erreur. Un réel occupe 5 octets (40 bits) en mémoire.
nom de variable réelle	:= nom
variable arithmétique	:= vara
vara	:= nom nom%
	toutes les variables occupent 7 octets en mémoire: 2 pour le nom, 5 pour la valeur réelle ou entière.
séparateur	:= ~ () = - + > < / , ; :
	Un nom ne doit pas être séparé d'un mot réservé qui le précède ou le suit par un de ces délimiteurs.
opérateur arithmétique	:= opa
opa	:= + - * / ^
opérateur arithmétique logique	:= opal
opal	:= AND OR = > < <> >< >= => <= =<
	NOT est volontairement absent de cette liste.
opérateur	:= op
op	:= opa opal
expression arithmétique	:= expra
expra	:= vara réel entier
	:= (expra)
	Si les parenthèses sont emboîtées à plus de 36 ?OUT OF MEMORY ERROR s'affiche (pas assez de mémoire).
	:= [+ - NOT] expra
	NOT est bien la fonction logique NON.
	:= expra op expra
dimension	:= (expra{ }, expra{ })
	La dimension maximale est de 89 mais elle sera limitée en pratique par la taille mémoire disponible. Expra doit être positive et est convertie en entier.
vara	:= vara dimension
expra	:= vara dimension
littéral	:= { } caractère { }
chaîne de caractères alphanumériques	:= chaîne

chaîne	:= "[{caractère}]" Une chaîne occupe 1 octet (8 bits) pour sa longueur, 2 octets pour son pointeur et 1 octet pour chaque caractère.
	:= "[{caractère}] RETURN" Ce type de chaîne ne peut apparaître qu'en fin de ligne.
chaîne nulle	:= ""
nom de la variable alphanumérique	:= nom §
variable alphanumérique	:= varc
varc	:= nom § nom § dimension Le pointeur de chaîne et le nom de la variable occupent tous deux 2 octets en mémoire. La longueur de la chaîne et chaque caractère la composant occupent 1 octet en mémoire.
opérateur de chaîne	:= opc
opc	:= +
expression de chaîne	:= exprc
exprc	:= varc chaîne := exprc opc exprc
opérateur logique de chaîne	:= oploc
oploc	:= = > >= > < <= < <> ><
expra	:= exprc oploc exprc
variable	:= var
var	:= vara varc
caractère de reconnaissance du B.E.V.F.	:=] Le crochet s'affichera quand B.E.V.F. est prêt à recevoir une autre commande.
expression	:= expr
expr	:= expra exprc
RESET	:= enfoncer la touche marquée RESET
ESC	:= enfoncer la touche marquée ESC
RETURN	:= enfoncer la touche marquée RETURN
CTRL	:= tenir enfoncée la touche CTRL pendant qu'une autre touche est enfoncée.
numéro de ligne	:= numligne

```

numligne      := {chiffre}
               Les numéros des lignes doivent être compris
               entre 0 et 63999 sinon le message ?SYNTAX ERROR
               (erreur de syntaxe) apparaîtra.

ligne         := numligne [{instruction:}] instruction RETURN
               Une ligne peut avoir jusqu'à 239 caractères.
               Cela comprend les espaces ajoutés par l'utili-
               sateur mais pas ceux que le B.E.V.F. ajoute
               pour formater ces lignes.

```

RÈGLES DE PRIORITÉ DANS LES EXPRESSIONS

Voici la liste des opérateurs entrant dans l'évaluation des expressions. Cette liste doit être lue verticalement, la priorité la plus forte étant la plus haute de la liste et la plus faible, la plus basse de la liste. Les opérateurs différents mais sur la même ligne ont même priorité. Pour les opérateurs de même priorité leur exécution se fait de gauche à droite dans l'instruction.

```

()
+ - NOT                (opérateurs ne nécessitant qu'un seul argument)
^
* /
+ -                (opérateurs ?????????? deux arguments)
> < >= <= => =< <> ><=
AND
OR

```

CONVERSIONS DES EXPRESSIONS

Si un entier et un réel sont présents dans un même calcul alors l'entier est converti en réel avant que le calcul ne s'effectue. Le résultat est converti soit en réel, soit en entier, selon le type de la dernière variable où il est assigné. Les fonctions travaillant sur un certain type (soit réel, soit entier) feront les conversions des arguments conformément au type auquel elles appartiennent. Impossible de mélanger des expressions arithmétiques et alphanumériques dans un calcul, il faut préalablement les convertir dans le type désiré.

MODES D'EXECUTION

- imm := mode immédiat en B.E.V.F. Concerne quelques instructions exécutables sans numéro de ligne et suivies d'une pression sur la touche **RETURN** qui exécute immédiatement l'instruction.
- pg := les instructions utilisées en mode programme doivent être écrites avec un numéro de ligne au début. Après avoir pressé la touche **RETURN** en fin de ligne. Le B.E.V.F. stocke la ligne de programme en mémoire pour un usage ultérieur. L'exécution se fait uniquement après avoir introduit la commande RUN.

Notes

CHAPITRE 3

COMMANDES DU SYSTEME ET DES UTILITAIRES

LOAD imm et pg
SAVE imm et pg

LOAD
SAVE

Ces instructions servent à charger (LOAD) ou à conserver (SAVE) un programme sur une cassette.

Il n'y a plus de curseur affiché ou autre signal lorsqu'on les utilise; l'utilisateur doit avoir son magnétophone qui se trouve, soit en lecture, soit en enregistrement lorsque SAVE ou LOAD sont utilisés. Il n'est pas vérifié si le magnétophone est bien réglé ou s'il est bien branché. Ces deux instructions émettent un "bip" qui signale le début et la fin du programme sur la cassette.

L'exécution d'un programme continue après avoir utilisé la commande SAVE, mais un LOAD détruit le programme en mémoire centrale pour le remplacer par celui de la cassette.

Seul RESET peut arrêter un LOAD ou un SAVE.

Si vous utilisez les mots réservés LOAD ou SAVE dans les premiers caractères du nom d'une variable, la commande s'adressant au mot réservé aura priorité sur le SYNTAX ERROR que vous devriez voir s'afficher.

L'instruction:

SAVERING = 5 provoque un essai de sauvegarde sur cassette de la part de B.E.V.F., du programme en mémoire.

Pour arrêter, attendre un second "bip" (avec le message SYNTAX ERR) ou bien appuyer sur la touche **RESET**.

NEW imm et pg
NEW

Pas de paramètres. Efface le programme de la mémoire ainsi que les variables.

RUN imm et pg
RUN [numligne]

Efface les anciennes valeurs des variables, des pointeurs en mémoire et commence l'exécution du programme au numéro de ligne (numligne) indiqué.

Ou si ce numéro est absent, à la ligne de programme dont le numéro est le plus petit. S'il n'y a pas de programme en mémoire, le contrôle de l'ordinateur est redonné à l'utilisateur.

Si RUN est utilisé en programme et qu'il n'existe pas de numéro de ligne correspondant au numéro derrière le RUN ou si ce numéro est négatif, le message: ?UNDEF'D STATEMENT ERROR

s'affiche (instruction inexistante). Si le numéro est supérieur à 63999 le message: ?SYNTAX ERROR s'affiche. On ne vous indique pas à quelle ligne s'est produite l'erreur.

En mode immédiat, ces messages deviennent:

? UNDEF'D STATEMENT ERROR IN XXXX

et

?SYNTAX ERROR IN XXXX

Ou XXXX est un numéro de ligne dont il ne faut pas tenir compte (en général supérieur à 65000). Si RUN est utilisé dans un programme en mode d'exécution immédiat, alors toute partie suivant ce programme est ignorée.

STOP	imm et pg
END	imm et pg
CTRL	imm seulement
C	
RESET	imm seulement
CONT	imm et pg

STOP
END
CTRL
C
RESET
CONT

STOP arrête un programme en cours d'exécution, rend le contrôle de l'ITT 2020 à l'utilisateur en imprimant le message BREAK IN numligne. Numligne est le numéro de ligne où se trouvait l'instruction STOP dans le programme.

END arrête l'exécution du programme, rend le contrôle à l'utilisateur mais n'affiche pas de message.

CTRL a le même effet que d'insérer STOP à la fin de l'instruction que le programme est en train d'exécuter. Cela vous permet d'arrêter un programme quand vous le voulez pendant le cours de l'exécution.

On peut aussi utiliser **CTRL** pour interrompre un LISTING. Il peut aussi interrompre un

INPUT à condition qu'il soit le premier caractère de INPUT. Tant que **RETURN** n'est pas enfoncé le programme est toujours en train d'exécuter l'INPUT.

RESET arrête tout programme B.E.V.F. dans n'importe quelle condition. Le programme n'est pas perdu mais seulement quelques pointeurs programme et pointeurs de files sont réinitialisés. Cette commande vous fait accéder au moniteur de l'ITT 2020 et le signe de reconnaissance du moniteur (*) s'affiche. (Sauf avec la ROM spéciale d'initialisation automatique fournie avec option).

Pour revenir au B.E.V.F. sans perdre votre programme, tapez **CTRL** puis **RETURN**.

CTRL
C

RETURN

Si l'exécution d'un programme a été interrompue par un STOP, un END ou un **CTRL**, la commande CONT fait recommencer le programme à L'INSTRUCTION (et non au numéro de ligne) suivante.

Rien n'est effacé. Si le programme n'a pas été arrêté, l'instruction CONT n'a aucun effet. Après un **RESET**, **CTRL**, **RETURN** l'instruction CONT risque de ne pas marcher, en effet le **RESET** aura effacé des pointeurs programmes et pointeurs de files.

Si vous interrompez le programme au cours d'une instruction INPUT et que vous essayez de le CONTinuer ensuite, un ?SYNTAX ERROR IN numligne s'affichera ou numligne sera la ligne où le programme avait été interrompu.

Exécuter la commande CONT génèrera le message d'erreur ?CAN'T CONTINUE ERROR (Impossible de continuer, erreur), si après avoir arrêté le programme, l'utilisateur:

- a - modifie ou supprime une ligne du programme
- b - tape une commande provoquant un message d'erreur.

Pendant il y aura CONTinuation sans erreurs si les variables sont modifiées en mode immédiat et sans qu'aucun message d'erreur ne s'affiche.



Si l'instruction DEL est utilisée dans un programme, les lignes spécifiées sont détruites, mais l'exécution du programme s'arrête. Un essai d'utilisation de CONT provoquera le message d'erreur:

?CAN'T CONTINUE ERROR

Si CONT est utilisé en mode programme, l'exécution du programme s'arrête, mais l'ordinateur n'est rendu au contrôle de l'utilisateur que si celui-ci tape **CTRL**.

Tout essai de CONTinuation du programme sera alors refusé et le programme se re-stoppera là où il s'était déjà bloqué.

TRACE imm et pg
 NOTRACE imm et pg

TRACE sélectionne le mode de dépistage de l'ITT 2020, il provoque l'affichage de chaque numéro de ligne à l'exécution de chaque instruction de la ligne en cours. Quand votre programme affiche aussi du texte sur l'écran, le dépistage sur l'écran (TRACE) peut s'afficher sur vos textes. NOTRACE "débranche" la fonction de dépistage des lignes TRACE.

Une fois sélectionné TRACE n'est pas "débranché" par RUN, CLEAR, NEW, DEL ou RESET, mais **RESET** **CTRL** le "débranche" et détruit tout programme en mémoire centrale.

PEEK imm et pg
 PEEK (expra)

va chercher en mémoire centrale et récupère en DECIMAL la valeur de l'octet à l'adresse \expa\ . Vous trouverez à l'annexe J les exemples d'utilisation de PEEK.

POKE imm et pg
POKE expra 1, expra 2

POKE écrit dans la mémoire un octet, équivalent binaire de la valeur décimale \expra 2\, à l'adresse donnée par expra 1.

\expra 2\ doit être compris entre 0 et 255
\expra 1\ doit être compris entre -65535 et +65535.

Les réels sont convertis en entiers avant l'exécution de l'instruction.

Les valeurs hors des intervalles provoquent le message d'erreur: ?ILLEGAL QUANTITY ERROR (quantité illégale, erreur).

\expra 2\ sera effectivement écrit si l'adresse \expra 1\ correspond à une zone hardware de l'ITT 2020 où l'on peut physiquement écrire (RAM ou équipement de sortie appropriée).

\expra 2\ ne pourra s'écrire dans les ROM ou bien dans les mémoires des connecteurs d'entrée/sortie non utilisés.

Cela veut dire qu'en général \expra 1\ doit être compris entre 0 et un maximum déterminé par la configuration mémoire de l'appareil.

Par exemple, avec un ITT 2020 de 16K de mémoire, le maximum sera de 16384. Sur un ITT 2020 de 32K il serait de 32768 et sur une configuration de 48K de 49152.



Attention, beaucoup de cases mémoire ont une fonction nécessaire au B.E.V.F., écrire (POKE) des valeurs dans ces cases pourrait affecter le bon fonctionnement du B.E.V.F. et vous risqueriez de perdre votre programme.

WAIT imm et pg
WAIT expra 1, expra 2 [, expra 3]

Permet à l'utilisateur d'insérer une pause conditionnelle dans un programme. Seul **RESET** peut interrompre un WAIT. \expra 1\ est l'adresse d'une case mémoire, et doit être compris entre -65535 et +65535 pour éviter l'erreur: ?ILLEGAL QUANTITY ERROR (quantité illégale, erreur) qui s'afficherait. En pratique \expra 1\ doit se limiter aux adresses de la zone mémoire disponible, c'est-à-dire de 0 jusqu'au maximum fixé par HIMEM: dans votre ordinateur. (Se référer à HIMEM: et POKE pour plus de détails).
Les adresses peuvent être indifféremment positives ou négatives.

\expra 2\ et \expra 3\ doivent être compris entre 0 et 255 en décimal. A l'exécution du WAIT ces valeurs converties en binaire peuvent s'échelonner entre 00000000 et 11111111. Si \expra 1\ et \expra 2\ sont seuls spécifiés alors chacun des 8 bits du contenu de \expra 1\ est "intersecté" (fonction AND) avec l'équivalent binaire de \expra 2\.

Cela donne pour chaque bit un zéro sauf si les 2 bits étaient à 1 avant l'intersection.

Si le résultat est 00000000 alors l'intersection est recommencée.

Si le résultat est non nul (ce qui veut dire qu'il y a au moins un bit à 1 après l'intersection) le programme continue alors son exécution à l'instruction suivante.

WAIT expra 1,7

mettra le programme en pause jusqu'à ce qu'un des 3 bits les plus à droite du contenu de l'adresse \expra 1\ soit à 1.

WAIT expra 1,0 met le programme en pause ad vitam aeternam!

Si les trois paramètres sont spécifiés, le WAIT s'exécute comme suit:

• en premier, chaque bit du contenu de l'adresse \expra 1\ est réuni exclusivement (fonction ou exclusive) avec le bit correspondant de \expra 3\.

Un bit à 1 dans \expra 3\ donne la valeur inverse du bit correspondant du contenu de l'adresse \expra 1\ (un 1 devient 0, un 0 devient 1). Un bit à 0 dans \expra 3\ fait que le bit correspondant dans le contenu de l'adresse \expra 1\ garde sa valeur initiale d'avant la réunion exclusive.

Si \expra 3\ vaut zéro alors le OU exclusif est sans effet.

• En second, chaque résultat est intersecté logiquement avec la valeur binaire de \expra 2\.

Si le résultat final est 8 zéros, le test recommence.

Si un résultat final est non nul, l'exécution du programme continue à l'instruction suivante. L'instruction WAIT sert en fait à tester le contenu de l'adresse \expra 1\ et voir si CERTAINS des bits sont à 1 et lesquels, ou inversement lesquels sont à zéro.

Vous pouvez choisir les bits qui vous intéressent dans le contenu de l'adresse \expra 1\ en fixant les bits, de l'équivalent binaire de \expra 2\, correspondant à 1 s'ils vous intéressent, à 0 sinon.

Chaque bit de la valeur binaire de \expra 3\ indique que vous attendez (WAIT) la même valeur du bit correspondant au contenu de l'adresse \expra 1\ et 0 dans le contenu de l'adresse \expra 1\ et 0 veut dire que le bit doit être à 1 dans le contenu de l'adresse \expra 1\.

Si un des bits contenus à l'adresse \expra 1\ que vous avez considéré comme intéressant en mettant à 1 le bit correspondant de \expra 2\ égalise l'état spécifié pour ce bit (par le bit correspondant de \expra 3\) la pause est terminée (WAIT).

Si \expra 3\ n'apparaît pas dans l'instruction, il est considéré comme nul.

Exemples:

WAIT, expra 1, 255, 0	pause jusqu'à ce qu'au moins un des 8 bits du contenu de l'adresse \expra 1\ soit à 1.
WAIT expra 1, 255	indentique à ci-dessus.
WAIT expra 1, 255, 255	pause jusqu'à ce qu'au moins un des 8 bits du contenu de l'adresse \expra 1\ soit à 0.
WAIT expra 1, 1, 1	pause jusqu'à ce que le bit le plus à droite du contenu de l'adresse \expra 1\ soit à 0 sans tenir compte de l'état des autres bits.
WAIT expra 1, 3, 2	pause jusqu'à ce que le bit le plus à droite du contenu de l'adresse \expra 1\ soit à 1 ou bien que son voisin soit à 0 ou bien les deux.

Programme qui illustre une pause en attendant un caractère dont le code ASCII (voir annexe K) est pair:

```

100 POKE -16368,0 : REM MET A ZERO LE STROBE (INDICATEUR DE PRESSION DE TOUCHE)
105 REM UNE PAUSE POUR ATTENDRE QUE LE STROBE SOIT A 1 PAR PRESSION DE TOUCHE
110 WAIT -16384,128 : REM ATTEND JUSQU'A CE QUE LE BIT DE POIDS FORT SOIT A 1
115 REM ENCORE UNE PAUSE JUSQU'A CE QUE LE CODE ASCII TAPE SOIT PAIR
120 WAIT -16384, 1, 1 : REM ATTEND LE BIT DE POIDS FAIBLE A 0
130 PRINT "PAIR" : PRINT
140 GOTO 100

```


CALL imm et pg
CALL expra

Provoque l'appel d'un sous-programme en langage machine débutant à l'adresse décimale spécifiée par \expra\.

\expra\ doit être compris entre -65535 et +65535 sinon le message: ?ILLEGAL QUANTITY ERROR (quantité illégale, erreur) apparaît. En pratique \expra\ se limite à la zone de mémoire disponible c'est-à-dire entre 0 et la valeur maximale fixée par HIMEM: dans votre ordinateur.

Se référer à HIMEM: et POKE pour plus de détails.

Le signe des adresses n'a pas d'importance.

CALL -936 et CALL 64600 sont identiques.

L'usage de CALL est employé à l'annexe J.

HIMEM: imm et pg
HIMEM: expra

Fixe la valeur de la plus haute case mémoire disponible pour un programme BASIC, variables incluses. Utilisé pour protéger une zone de mémoire pour des données, pour des graphiques ou des programmes en langage machine.

\expra\ doit être compris entre -65535 et 65535 pour éviter le message: ?ILLEGAL QUANTITY ERROR. Cependant la valeur de HIMEM: ne peut être fantaisiste et doit rester dans les limites de la configuration mémoire disponible sur votre ITT 2020.

En général, la valeur maximale de \expra\ est déterminée par la capacité des RAM.

Pour un ITT 2020 de 16K mémoire \expra\ devra être de 16384 ou moins
Pour un ITT 2020 de 32K mémoire \expra\ devra être de 32768 ou moins
Pour un ITT 2020 de 48K mémoire \expra\ devra être de 49152 ou moins.

B.E.V.F. fixe automatiquement le HIMEM: à la plus haute case mémoire disponible sur l'ordinateur de l'utilisateur, à condition que ce soit le B.E.V.F. qui soit appelé en premier comme langage après l'allumage.

La valeur du HIMEM: est conservé dans les cases mémoire 115 et 116 (décimal). Pour connaître la valeur de HIMEM:, tapez:

```
PRINT PEEK (116) * 256 + PEEK (115)
```

Si le HIMEM: fixe la plus haute adresse à une valeur plus basse que le LOMEM:, ou qui ne laisse pas suffisamment de place au programme pour s'exécuter, le message: ?OUT OF MEMORY ERROR (plus de mémoire, erreur) s'affiche.

\expra\ peut varier de 0 à 65535, ou équivalent: de -65535 à -1. L'usage des valeurs positives ou négatives est indifférent.

CTRL
C

, CLEAR, RUN, DEL, NEW, changer de programme, ajouter des instructions ne changent pas la valeur de HIMEM:

Le HIMEM: se réinitialise en tapant

RESET

CTRL
B

RETURN

 (le programme est aussi effacé).

LOMEM: imm et pg
LOMEM: expra

Fixe l'adresse de la plus basse case mémoire disponible pour un programme BASIC. C'est en général l'adresse d'implantation de la première variable BASIC. B.E.V.F. fixe le LOMEM: automatiquement à la fin du programme en mémoire avant l'exécution de celui-ci. Cette commande protège les variables des données haute résolution sur les ordinateurs disposant d'une large place mémoire.

\expra\ doit être compris entre -65535 et 65535 pour éviter:

?ILLEGAL QUANTITY ERROR (quantité illégale, erreur) qui s'afficherait sinon. Si le LOMEM: est fixé plus grand que le HIMEM: alors le message: ?OUT OF MEMORY ERROR (plus de mémoire, erreur) s'affiche. Cela indique que le LOMEM: doit FORCEMENT être inférieur à la valeur du HIMEM: fixé. (Cf. HIMEM:)

Si le LOMEM: est fixé inférieur à la plus haute case mémoire réservée au système (plus tout programme en mémoire) le message ?OUT OF MEMORY ERROR s'affichera de nouveau. Cela impose la limite inférieure de \expra\ à 2051 pour B.E.V.F.

LOMEM: est réinitialisé par NEW, DEL, et par ajout ou changement d'une ligne de programme. LOMEM: est aussi réinitialisé par

CTRL
B

, qui détruit aussi le programme en mémoire.

Il n'est pas modifié par: RUN,

RESET

CTRL
C

RETURN

, ou RESET ØG RETURN.

La valeur du LOMEM: est conservée dans les cases mémoire 106 et 105 (décimal). Pour connaître la valeur du LOMEM:, tapez:

```
PRINT PEEK (106)*256 + PEEK (105)
```

Une fois fixé LOMEM:, s'il n'a pas été modifié par les commandes vues précédemment, on peut le modifier UNIQUEMENT si on l'augmente. Une tentative de diminuer la valeur de LOMEM: se soldera par: ?OUT OF MEMORY ERROR

Modifier le LOMEM: pendant l'exécution d'un programme risque de rendre des parties de programme indisponibles; le programme ne pourra plus s'exécuter correctement.

Les adresses peuvent être indifféremment positives ou négatives.

USR imm et pg
USR (expra)

Cette fonction communique une expression (\expra\) à un sous-programme en langage machine. L'argument expra est calculé puis placé dans l'accumulateur à virgule flottante (adresse de § 9D jusqu'à § A3, la notation § signifie hexadécimal), puis un JSR à l'adresse §0A est exécuté. Les adresses § 0A, § 0B, § 0C doivent contenir un JMP à un sous-programme en langage machine. La valeur de retour est placée dans l'accumulateur à virgule flottante.

Pour obtenir un entier sur 2 octets de la valeur contenue dans l'accumulateur à virgule flottante, votre sous-programme devra contenir et exécuter un JSR § E10C.

Après le retour, la valeur de l'entier sera placée aux adresses § A0 (octet de poids fort) et § A1 (octet de poids faible).

Pour convertir un entier en réel pour que la fonction puisse retrouver sa valeur, placez

la valeur entière dans le registre A (octet de poids fort) et Y (octet de poids faible).
Puis exécutez un JSR § E2F2.

Après le retour, la valeur réelle sera dans l'accumulateur à virgule flottante.
Pour revenir en B.E.V.F., n'oubliez pas un RTS.

Voici un petit programme illustrant l'emploi de la fonction USR:

```
] 

|       |
|-------|
| RESET |
|-------|

  
* 0A : 4C 00 03 

|        |
|--------|
| RETURN |
|--------|

  
* 0300 : 60 

|        |
|--------|
| RETURN |
|--------|

  
* 

|      |
|------|
| CTRL |
| C    |



|        |
|--------|
| RETURN |
|--------|

  
] PRINT USR (8) * 3
```

24

A l'adresse § 0A nous avons placé un JMP (code 4C) à l'adresse 300. A l'adresse 300, nous avons mis un RTS (code 60).

Après avoir récupéré B.E.V.F., à l'exécution du USR (8), la valeur 8 est placée dans l'accumulateur, le moniteur fait un JSR à l'adresse § 0A où il trouve un JMP§ 300. Puis en 300 il trouve un RTS qui le renvoie au B.E.V.F.

La valeur retrouvée était bien la valeur originale de 8 qui multipliée par 3 par B.E.V.F. affiche 24.

Notes

CHAPITRE 4

COMMANDES D'EDITION ET DE FORMATS D'AFFICHAGE

```
LIST          imm et pg
LIST          [numligne 1] [-numligne 2]
LIST          [numligne 1] [,numligne 2]
```

Si numligne 1 et numligne 2 ne sont pas spécifiés, c'est tout le programme qui s'affiche sur l'écran. Si numligne 1 est présent, sans séparateur, ou si numligne 1 = numligne 2 alors seulement la ligne numligne 1 s'affiche.

Si numligne 1 et un séparateur sont présents, alors le programme se LISTe de numligne 1 à la fin.

Si numligne 1, un séparateur, numligne 2 sont présents, le LISTage du programme s'affiche à partir de numligne 1 jusqu'à numligne 2 comprise.

Si un séparateur et numligne 2 sont présents, le programme s'affiche du début jusqu'à la ligne numligne 2 comprise.

Si, à votre demande de LISTer plusieurs lignes, la ligne du programme de numéro num- ligne 1 n'existe pas, le LISTing commence au premier numéro de ligne supérieure à numligne 1 existant dans le programme.

Inversement, si la ligne numligne 2 n'existe pas dans le programme, le LISTing s'arrête à la dernière ligne dont le numéro est inférieur à numligne 2.

Présentées comme ceci, elles LISTent tout le programme:

```
LIST Ø LIST [,|-] Ø LIST Ø [,|-] Ø
LIST numligne, Ø      LIST le programme de numligne jusqu'à la fin.
```

LIST, Q

LISTe le programme en entier et affiche:

```
?SYNTAX ERROR
```

B.E.V.F. formate vos lignes de programme avant de les stocker en mémoire, il enlève les espaces qui ne sont pas nécessaires. Au LISTing, B.E.V.F. restitue les lignes en ajoutant des espaces selon ses propres règles.

Par exemple:

```
1Ø C = +5/-6 : B = -5
```

devient:

```
1Ø C = +5 : -6 : B = -5
```

quand on LISTe.

LIST utilise des largeurs de lignes variables et un formatage variable à l'affichage. Cela peut poser des problèmes si vous désirez éditer ou modifier des instructions LISTées. Pour forcer le B.E.V.F. à abandonner son formatage qui ajoute des espaces, nettoyez l'écran et diminuez la largeur de la fenêtre de l'écran à 33 (maximum):

HOME
POKE 33, 33



Le B.E.V.F. traite des lignes de 239 caractères PUIS LIST ajoute des espaces. Vous pouvez gagner des caractères en tapant vos lignes sans y introduire aucun espace (LIST les ajoutera lui-même).

Si vous essayez de recopier une ligne qui faisait 239 caractères et ne contenait pas d'espaces, le LIST ajoutant des espaces, vous perdriez des informations en tentant de la recopier, car le B.E.V.F. comptabiliserait alors les espaces dans 239 caractères autorisés.

Un LISTing s'arrête par un

CTRL
C

DEL imm et pg
DEL numligne 1, numligne 2

Efface du programme les lignes de programmes comprises entre les numéros numligne 1 et numligne 2 inclus.


Si numligne 1 n'existe pas dans le programme, la première ligne supprimée est la première ligne dont le numéro est supérieur à numligne 1.

Inversement si numligne 2 n'existe pas dans le programme, la dernière ligne supprimée est la dernière ligne dont le numéro est inférieur à numligne 2.

Si vous n'utilisez pas le format usuel, voilà ce qui risque de se passer:

SYNTAXE

RESULTAT

DEL	?SYNTAX ERROR (erreur de syntaxe)
DE,	?SYNTAX ERROR
DEL, b	?SYNTAX ERROR
DEL -a [,b]	?SYNTAX ERROR
DEL Ø, b	efface la ligne Ø sans toucher à aucune autre.
DEL l, -b	ignorée même si la plus petite ligne du programme est Ø
DEL a, -b	?SYNTAX ERROR si (a) est plus grand que le plus petit numéro de la ligne du programme, à moins que le plus petit numéro de la ligne du programme soit Ø et que (a) vaut 1.
DEL a, -b	ignorée si (a) est différent de zéro et que le seul numéro de ligne du programme soit Ø.
 DEL a, -b	ignorée si (a) est différent de zéro et si (a) est plus petit ou égal au plus petit numéro de ligne du programme.
DEL a ,	ignorée.
DEL a, b	ignorée si (a) est différent de zéro et (a) plus grand que (b)



En utilisant DEL en mode programme il s'exécute selon les règles ci-dessus puis ARRETE le programme. CONT NE FONCTIONNERA PLUS ensuite.

```
REM      imm et pg
REM      {caractère|"}

```

Cela permet d'insérer du texte dans un programme. Tous les caractères, y compris les séparateurs d'instructions et les espaces peuvent être inclus dans REM. A l'exécution le B.E.V.F. ignore tous les caractères suivants le REM. Une REMarque doit être terminée par un RETURN.

Quand les REMarques sont LISTées, B.E.V.F. insère un espace juste après REM, sans tenir compte des espaces tapés après le REM par l'utilisateur.

```
VTAB      imm et pg
VTAB      expra

```

Déplace le curseur à la ligne spécifiée par \expra\ . Le haut de l'écran est la ligne 1, le bas 24.

Cette instruction permet des déplacements haut-bas mais jamais droite-gauche. (Ex:]VTAB 10 : PRINT 123 RETURN).

\expra\ doit être compris entre 1 et 24 pour éviter que ?ILLEGAL QUANTITY ERROR (quantité illégale, erreur) s'affiche.

Les déplacements sont absolus par rapport au haut de l'écran. Ils ignorent la taille de la fenêtre de texte. En mode graphique, VTAB peut déplacer le curseur dans la zone graphique de l'écran.

Si un VTAB déplace le curseur sur une ligne en-dessous de la fenêtre de texte, tous les ordres d'affichage qui suivent seront faits sur cette ligne.

```
HTAB      imm et pg
HTAB      expra

```

Sachant qu'une ligne où le curseur est placé à 255 positions différentes. Sans tenir compte de la largeur fixée de la fenêtre de texte, les positions de 1 à 40 sont sur la même ligne, de 41 à 80 sur la ligne au-dessous, etc.

HTAB déplace le curseur de \expra\ positions par rapport au bord gauche de la ligne où le curseur est placé. (Ex:]HTAB 10 : PRINT 123 RETURN)

Les déplacements HTAB sont relatifs à la marge gauche de la fenêtre de l'écran, mais indépendants de la largeur de la ligne.

HTAB peut déplacer le curseur hors de la fenêtre de texte, mais seul un caractère pourra s'afficher hors de la fenêtre.

HTAB 1 place le curseur sur le bord gauche de l'écran.



HTAB 0 déplace le curseur à la position 256. Si \expra\ est négatif ou supérieur à 255, le message ?ILLEGAL QUANTITY ERROR (quantité illégale, erreur) s'affiche.

Notez bien que les structures de HTAB et VTAB ne sont pas identiques. En effet, un HTAB derrière le bord droit de l'écran ne cause pas le ?ILLEGAL QUANTITY ERROR, mais provoque un saut du curseur à la ligne suivante à la position:

$((\text{expra} - 1) \text{MODULO } 4\emptyset) + 1$

TAB imm et pg
TAB (expra)

TAB DOIT être utilisé avec une instruction PRINT. Et `expra` doit être entre parenthèses. TAB déplace le curseur de `\expra\` positions par rapport au bord gauche de la fenêtre de texte seulement si `\expra\` est plus grand que la valeur de la position du curseur relativement au bord gauche de la fenêtre de texte.

Si `\expra\` est inférieur à la valeur de la position du curseur, le curseur n'est pas déplacé, donc TAB ne déplace jamais le curseur vers la gauche (utiliser alors HTAB).

Si TAB déplace le curseur derrière le bord droit de la fenêtre de texte, les déplacements se continuent alors sur la ligne suivante.



TAB (\emptyset) met le curseur en position 256.
`\expra\` doit être compris entre \emptyset et 255 ou le message:
?ILLEGAL QUANTITY ERROR (quantité illégale, erreur) s'affichera.

TAB est considéré comme un mot réservé seulement s'il est suivi par une parenthèse ouvrante.

POS imm et pg
POS (expr)

Donne la position horizontale du curseur sur l'écran par rapport au bord gauche de la fenêtre de texte. Au bord gauche, \emptyset est affiché. POS doit être utilisé avec un PRINT.

La valeur de l'expression (`expr`) est sans importance, et ne peut être illégale. Si `expr` est un jeu de caractères ne représentant pas un nom de variable, ces caractères doivent être entre guillemets.

REMARQUE: les positions pour HTAB et TAB sont numérotées à partir de 1, mais pour POS et SPC numérotées à partir de \emptyset .

Par conséquent:

PRINT TAB (23) ; POS (\emptyset) affiche 22, et
PRINT SPC (23) ; POS (\emptyset) affiche 23

SPC imm et pg
SPC (expra)

Doit être utilisé avec une instruction PRINT et `expra` doit être entre parenthèses. Laisse `\expra\` espaces entre le dernier texte affiché (ou par défaut la marge gauche de la fenêtre de texte) et le prochain à afficher. (Si la concaténation de SPC avec les ordres PRINT précédents s'est fait avec les points-virgules).

Exemple:

PRINT "BONJOUR" ; SPC (4) ; "AU REVOIR"

SPC (Ø) n'introduit pas d'espaces.

\expra\doit être compris entre Ø et 255 inclusivement, ou bien le message:

?ILLEGAL QUANTITY ERROR (quantité illégale, erreur) s'affiche.

Cependant, les concaténations peuvent se faire sous cette forme:

PRINT SPC (25Ø) SPC (139) SPC (255)

pour dégager de grands espaces.

Remarquez que si HTAB déplaçait le curseur relativement au bord gauche de la fenêtre de texte, SPC (expra) déplace le curseur relativement au dernier texte affiché.

Cette nouvelle position du curseur peut être n'importe où dans la fenêtre de texte, cela dépend de la position du dernier texte affiché.

Espacer plus loin que le bord droit de la fenêtre de texte provoque l'affichage à la ligne suivante. Quand vous affichez dans les champs de tabulation, l'espacement peut se faire dans un champ de tabulation ou à cheval sur deux champs de tabulation.

Si \expra\ est un réel, il est converti en entier, SPC est considéré comme un mot réservé seulement si le caractère qui suit SPC est une parenthèse ouvrante.

HOME imm et pg

HOME

Sans paramètres. Déplace le curseur en haut à gauche de la fenêtre de texte en nettoyant l'écran de tous les caractères à l'intérieur de la fenêtre.

Cette commande est identique à CALL -936 et



RETURN.

CLEAR imm et pg

CLEAR

Sans paramètres. Remet à zéro toutes les variables, les tableaux et les chaînes de caractères. Réinitialise les pointeurs et les piles.

FRE imm et pg

FRE (expr)

FRE donne la place mémoire (en octets) qui reste disponible à l'utilisateur.

Le résultat affiché peut souvent être supérieur à celui attendu, car le B.E.V.F. ne stocke qu'une fois les chaînes qui se dupliquent. C'est-à-dire que si A\$ = "BONJOUR" et B\$ = "BONJOUR", la chaîne BONJOUR ne sera stockée en mémoire qu'une fois.

Si le nombre d'octets libres est supérieur à 32767 FRE (expr) renvoie un nombre négatif et il suffit d'ajouter 65536 à ce nombre pour avoir la valeur positive du nombre d'octets libres.

FRE (expr) renvoie le nombre d'octets disponibles entre le dessous de la zone de stockage des chaînes de caractères et le dessus de la zone de stockage des tableaux et pointeurs de chaînes de caractères (Se référer à l'annexe I).

On peut fixer le HIMEM: aussi haut que 65535 mais s'il est fixé au-delà du maximum de RAM

(mémoire disponible) de votre ITT 2020. FRE risque de renvoyer une valeur aberrante et supérieure à la capacité mémoire de votre ordinateur (Pour plus de précisions, voir POKE et HIMEM:).

Quand le contenu d'une variable alphanumérique est modifié dans un programme (par exemple: A\$ = "CHAT" devient A\$ = "CHIEN") B.E.V.F. n'élimine pas CHAT mais réserve de la place pour CHIEN. Il en résulte un remplissage progressif de la mémoire, à partir du HIMEM: en descendant. Le B.E.V.F. fera automatiquement un "nettoyage maison" si jamais la zone de stockage des chaînes de caractères vient à empiéter sur celle des tableaux et pointeurs de chaînes. Mais si vous utilisez cet espace libre pour d'éventuels programmes en langage machine ou dessins haute résolution ils risquent d'être détruits.

Utiliser périodiquement une instruction du type:

```
X = FRE (0)
```

dans votre programme pour forcer le B.E.V.F. à effectuer le "nettoyage maison".

L'expression \expr\ peut être un nombre ou un nom de variable ou toute chaîne de caractères. (Elle ne peut comporter de noms illégaux).

```
FLASH      imm et pg
INVERSE    imm et pg
NORMAL     imm et pg
```

ou

```
FLASH
INVERSE
NORMAL
```

Les commandes servent à fixer le mode d'affichage du texte sur l'écran. Pas de paramètres et ces commandes n'affectent pas les caractères tapés au clavier, mais seulement les caractères affichés par l'ordinateur après un RETURN ou dans un programme. Les caractères déjà sur l'écran ne sont pas affectés non plus.

FLASH fixe le mode clignotant, c'est-à-dire que les caractères s'affichent alternativement blancs sur fond noir et noirs sur fond de ligne blanche.

INVERSE fixe le mode inversé sur les caractères; ils sont affichés en noir sur fond blanc.

NORMAL fixe le mode normal, c'est-à-dire caractères blancs sur fond noir (à l'entrée ou à la sortie de ces caractères).

```
SPEED      imm et pg
SPEED = expr
```

Fixe la vitesse à laquelle les caractères sont communiqués à l'écran ou aux entrées/sorties. La vitesse la plus lente est 0 et la plus rapide 255.

Si \expr\ est hors de ces limites le message: ?ILLEGAL QUANTITY ERROR (quantité illégale, erreur) s'affiche.

ESC	A	imm seulement	(pour l'édition)
ESC	B	imm seulement	"
ESC	C	imm seulement	"
ESC	D	imm seulement	"

La touche "ESC" du clavier utilisée avec les touches A, B, C ou D déplace le curseur: pour déplacer le curseur d'un espace, presser d'abord **ESC** PUIS ENSUITE en ayant relâché **ESC** taper **A**, **B**, **C** ou **D**.

COMMANDE		DEPLACEMENT DU CURSEUR
ESC	A	droite
ESC	B	gauche
ESC	C	bas
ESC	D	haut

Ces commandes permettent sans modifier les caractères affichés sur l'écran de déplacer le curseur. Elles n'affectent pas les programmes ou les lignes affichées. Pour modifier une ligne de programme, LISTEZ la ligne sur l'écran et en utilisant la touche **REPT** positionnez le curseur au tout début de la ligne à modifier (1er chiffre du numéro de ligne). Puis utiliser la touche **REPT** et la flèche à droite pour recopier les caractères de la ligne. Arrêtez-vous sur les caractères que vous désirez changer et changez-les. Si vous n'avez pas fait LISTER la ligne ne recopiez pas le caractère () du début de ligne. Pressez enfin **RETURN** pour stocker la ligne ou la faire exécuter.

Répétitions imm seulement (édition)

La touche de répétition est la touche notée **REPT**. Si vous appuyez sur une touche puis sans relâcher cette touche sur **REPT**, le caractère de la touche sera répété.

La première fois que vous enfoncez la touche **REPT** c'est le dernier caractère entré qui se répète.

Flèche à droite imm seulement (édition)
 Flèche à gauche imm seulement (édition)

La flèche à droite déplace le curseur à droite. Pendant le déplacement du curseur, chaque caractère rencontré sur l'écran est recopié dans la mémoire de l'ordinateur exactement comme si vous l'aviez tapé. On utilise la flèche à droite et la répétition pour recopier une ligne où il n'y a que des changements mineurs à faire.

La flèche à gauche déplace le curseur vers la gauche. A chaque fois que le curseur se déplace à gauche, le caractère traversé s'efface de la mémoire de l'ordinateur et de la ligne sur laquelle vous vous déplacez. Rien n'est changé sur l'écran.



A moins que vous soyez en train de taper une ligne et que vous n'avez pas encore appuyé sur **RETURN**, et si la flèche à gauche n'a pas de caractères à effacer sur la ligne: des | apparaîtront sur la colonne Ø de l'écran et en relâchant la touche le curseur réapparaîtra. C'est pourquoi le curseur ne peut être placé en colonne Ø à l'aide de la flèche.

Pour les simples déplacements, sans effacement, ou recopie: voyez la touche **ESC**.

CTRL
X

 imm seulement

Indique au B.E.V.F. d'ignorer la ligne que vous étiez certain de taper, sans pour cela effacer le contenu de l'ancienne ligne de même numéro.

Un \ s'affiche à la fin de la ligne à ignorer et le curseur réapparaît sur la ligne du dessous.

Cette commande peut être utilisée comme réponse à une instruction INPUT.

Notes

CHAPITRE 5

TABLEAUX

ET CHAINES DE CARACTERES

```
DIM      imm et pg
DIM      var dimension [ }, var dimension{ }
```

Quand une instruction DIM s'exécute elle fixe la place mémoire à réserver pour le tableau de nom var. Deux octets sont utilisés en mémoire pour conserver le nom de la variable de tableau, un pour le nombre de dimensions, et deux pour chaque dimension. Comme nous avons vu au chapitre 1, la place donnée aux éléments des tableaux dépend du type de tableau.

L'index varie de 0 à \dimension\.

Le nombre d'éléments d'un tableau à n dimensions est:

$(\backslash\text{dimension } 1\backslash + 1) * (\backslash\text{dimension } 2\backslash + 1) * \dots * (\backslash\text{dimension } n\backslash + 1)$

donc un tableau: DIM ALP (4, 5, 3) contiendra $5 * 6 * 4 = 120$ éléments.

Le nombre de dimensions maximales d'un tableau est de 88, même si chaque dimension contient un seul élément.

DIM A (0, 0....., 0) avec 89 zéros provoquera un ?OUT OF MEMORY ERROR (plus de mémoire, erreur) alors que DIMA (0, 0....., 0) avec 88 zéros ne le provoquera pas.

Cependant en pratique, la taille des tableaux est souvent limitée par la taille mémoire disponible. Chaque élément entier d'un tableau occupe 2 octets en mémoire. Chaque élément réel occupe 5 octets en mémoire. Chaque élément d'un tableau de chaînes de caractères occupe 3 octets, 2 pour le pointeur, 1 pour la longueur de la chaîne, stocké comme un élément de tableau entier quand le tableau est dimensionné.

Comme le programme stocke les chaînes dans la mémoire, elles occupent un octet par caractère.

Si un élément de tableau est utilisé dans un programme sans que la dimension ait été déclarée, B.E.V.F. assigne un nombre d'index possible de 10 maximum pour chaque dimension de l'élément.

Utiliser une variable dont l'index est supérieur au maximum fixé, ou si la variable appelle un nombre de dimensions supérieures à celui déclaré, provoquera le message:

?BAD SUBSCRIPT ERROR (erreur d'index) sur l'écran.

Si un programme DIMensionne un tableau dont la DIMension avait déjà été déclarée, le message: ?REDIM'D ARRAY ERROR (erreur de redimensionnement) apparaîtra.

Les variables alphanumériques n'ont pas besoin d'être dimensionnées, elles grandissent et diminuent au besoin. L'instruction:

```
WARD$ (5) = "ABCDE"
```

crée une chaîne de longueur 5. L'instruction:

WARD\$ (5) = " "

libère la place allouée à l'ancienne chaîne WARD\$ (5). Une variable alphanumérique peut contenir au maximum 255 caractères.

Les éléments de tableaux sont mis à zéro quand RUN ou CLEAR sont exécutés.

LEN imm et pg
LEN (exprc)

Cette fonction donne le nombre de caractères d'une chaîne, entre 0 et 255. Si exprc est une concaténation de chaînes et que la longueur totale est supérieure à 255, le message: ?OVERFLOW ERROR (dépassement de capacité, erreur) s'affiche.

STR\$ imm et pg
STR\$ (expra)

Convertit \expra\ en une chaîne de caractères représentant sa valeur. \expra\ est évaluée avant la conversion.

STR\$ (100 000 000 000) donne 1E + 11.

Si expra dépasse la limite des réels, le message ?OVERFLOW ERROR (dépassement de capacité, erreur) s'affiche alors.

VAL imm et pg
VAL (exprc)

Cette fonction transforme une chaîne de caractères en un entier ou un réel et affiche la valeur du nombre.

Le premier caractère de la chaîne doit être, ou bien un chiffre, ou bien un signe (+ -), ou un caractère appartenant à l'alphabet utilisé pour les nombres (E| |.), sinon c'est le zéro qui s'affiche.

Chaque caractère de la chaîne est examiné comme le premier. Et le premier caractère rencontré qui n'appartienne pas aux nombres est ignoré ainsi que TOUS les caractères qui suivent.

C'est à ce moment que la chaîne est évaluée en réel ou en entier.

Si la concaténation de chaînes fait que leur longueur dépasse 255 caractères et que vous essayez de l'évaluer, vous obtiendrez le message d'erreur: ?STRING TOO LONG ERROR (chaîne trop longue, erreur).

Si la valeur absolue du nombre transformé est plus grande que 1E 38 ou si le nombre contient plus de 38 chiffres (en incluant les zéros inutiles) le message: ?OVERFLOW ERROR (dépassement de capacité, erreur) s'affichera.

CHR\$ imm et pg
CHR\$ (expra)

C'est une fonction qui donne le caractère ASCII correspondant à la valeur de \expra\.
\expra\ doit être entre 0 et 255 sinon le message: ?ILLEGAL QUANTITY ERROR (quantité illégale, erreur) s'affiche.

Les réels sont convertis en entiers.

ASC imm et pg
ASC (exprc)

Cette fonction renvoie le code ASCII du premier caractère de \exprc\. Les codes ASCII entre 96 et 255 génèrent des caractères semblables à ceux entre 0 et 95. CHR% (65) affiche A et CHR% (193) aussi, mais le B.E.V.F. ne considère pas qu'il s'agisse des mêmes caractères dans des tests logiques sur les chaînes.

Si l'argument de ASC est une chaîne, elle doit être écrite entre guillemets. La chaîne elle-même ne doit pas contenir de guillemets. Si la chaîne est nulle, le message: ?ILLEGAL QUANTITY ERROR (quantité illégale, erreur) s'affiche.



Essayer la fonction ASC avec

CTRL
C

 comme argument provoquera le message d'erreur: ?SYNTAX ERROR (erreur de syntaxe).

LEFT% imm et pg
LEFT% (exprc, expra)

Cette fonction renvoie les \expra\ premiers caractères les plus à gauche de \exprc\.

```
PRINT LEFT% ("ITT 2020", 3)
ITT
```

Si \expra\ < 1 ou \expra\ > 255 le message:

?ILLEGAL QUANTITY ERROR (quantité illégale, erreur) s'affiche. Si \expra\ est un réel, il est converti en entier.

Si \expra\ > LEN (exprc) seuls les caractères constituant la chaîne sont affichés, le reste est ignoré.

Si le % est oublié du nom de la commande, B.E.V.F. considère LEFT comme une variable arithmétique et le message:

?TYPE MISMATCH ERROR (erreur de type) s'affiche.

RIGHT% imm et pg
RIGHT% (exprc, expra)

Cette fonction renvoie les \expra\ derniers (les plus à droite) caractères de \exprc\:

```
PRINT RIGHT% ("BONJOUR" + "NEE", 7)
JOURNEE
```

Aucune partie de la commande ne peut être omise.

Si \expra\ >= LEN (exprc) la fonction RIGHT% renvoie alors toute la chaîne.

Le message:

?ILLEGAL QUANTITY ERROR (quantité illégale, erreur) s'affiche si \expra\ < 1 ou \expra\ > 255.

RIGHT% (exprc, expra) = MID% (exprc, LEN (exptc) + 1 - \expra\)

Si le § est oublié de la commande, B.E.V.F. considère RIGHT comme une variable arithmétique et le message: ?TYPE MISMATCH ERROR (erreur de type) s'affiche.

```
MID§      imm et pg
MID§      (exprc, expra 1 [, expra 2])
```

MID§ avec deux arguments renvoie la sous-chaîne qui commence au \expra\ième caractère et finit au dernier caractère de \exprc\.

```
PRINT MID§ ("ITT 2Ø2Ø", 3)
2Ø2Ø
```

MID§ (exprc, expra) = RIGHT§ (exprc, LEN (exprc) + 1 - \expra\)
MID§ utilisé avec 3 arguments renvoie \expra 2\ caractères de \exprc\ à partir du \expra 1\ ième caractère.

```
PRINT MID§ ("ITT 2Ø2Ø", 2, 2)
TT
```

Si \expra 1\ > LEN (exprc) , MID§ renvoie alors la chaîne nulle.
Si \expra 1\ + \expra 2\ dépassent la longueur de \exprc\ (ou dépasse 255 maximum de longueur pour une chaîne), il n'y a pas d'erreur.

MID§ (A§, 255, 255) renvoie un caractère si LEN (A§) = 255 sinon la fonction renvoie la chaîne nulle.

Si \expra 1\ ou \expra 2\ ne sont pas compris entre 1 et 255, le message: ?ILLEGAL QUANTITY ERROR (quantité illégale, erreur) s'affiche.

Si le § est oublié de la commande, B.E.V.F. considère MID comme variable arithmétique et le message: ?TYPE MISMATCH ERROR (erreur de type) s'affiche.

```
STORE      imm et pg
RECALL     imm et pg
STORE      vara
RECALL     vara
```

Cette commande conserve et rappelle des tableaux chargés sur une cassette.
Le nom des tableaux n'est pas stocké, donc un tableau peut être relu en utilisant un autre nom que celui sous lequel le tableau avait été écrit (STORE).

Si, par exemple, un tableau dimensionné par DIMA (5, 5, 5) est écrit (STORE) on peut le relire (RECALL) en le plaçant dans un tableau dimensionné par DIM B (5, 5, 5).

Il faut strictement respecter les dimensions utilisées lors de la sauvegarde (STORE) car sinon cela entraînerait des désordres de nombres, des zéros supplémentaires ou l'erreur: ?OUT OF MEMORY ERROR (plus de mémoire, erreur).

En général le message ?OUT OF MEMORY ERROR s'affiche si le nombre total d'éléments prévu pour recevoir le tableau venant de la cassette est insuffisant.

```
DIM A (5, 5, 5)
STORE A
écrit 6 * 6 * 6 éléments sur la cassette.
```

```
DIM B (5, 25)
```


RECALL B

causera le message:

ERR (erreur)

et embrouillera les nombres du tableau B, mais le programme continuera son exécution.

Si vous rechargez par:

DIM B (5, 25)

RECALL B

le message:

?OUT OF MEMORY ERROR

s'affichera et le programme arrêtera son exécution. Le tableau B étant déclaré pour contenir 6 * 26 éléments, c'est-à-dire moins d'éléments que le tableau A n'en contenait.

Si le tableau rappelé (RECALL) a le même nombre de dimensions (DIM A (5, 5, 5) déclare un tableau 3 dimensions, chacun de 6 éléments) que le tableau qui fut conservé (STORE).

Le nombre d'éléments internes de chaque dimension du tableau rappelé (RECALL) peut être plus grand que celui du tableau conservé (STORE).

Cependant les nombres seront dans le désordre à moins que le nombre d'éléments de la DERNIERE dimension du tableau rappelé soit plus important que le nombre d'éléments de la dernière dimension du tableau conservé. Dans tous les cas, vous trouverez des 0 dans les éléments supplémentaires du tableau rappelé, mais simplement dans le dernier cas, vous trouverez des zéros où vous les attendiez. Après avoir conservé un tableau par:

DIM A (5, 5, 5)

STORE A

vous verrez que:

DIM B (10, 5, 5)

RECALL B

ou bien:

DIM B (5, 10, 5)

RECALL B

remplirons tous les deux le tableau B, avec les éléments de A mais dans le DESORDRE.

Mais si vous aviez fait:

DIM B (5, 5, 10)

RECALL B

cela aurait bien marché, et mis des 0 dans les éléments supplémentaires. Nous venons de voir 2 règles concernant STORE et RECALL avec un nombre de dimensions similaires.

- 1 - seule la dernière dimension du tableau rappelé (RECALL) peut être plus large que la dernière dimension du tableau conservé (STORE).
- 2 - le nombre total d'éléments rappelé doit être au moins égal au nombre d'éléments du tableau conservé.

Si vous respectez la règle 2 et si la règle 1 est respectée pour les dimensions communes aux deux tableaux, alors vous pouvez rappeler (RECALL) un tableau avec plus de dimensions que celui qui avait été conservé (STORE). Un message d'erreur (ERR) s'affiche mais le programme continue.

DIM B (5, 5, 5, 5)

RECALL B

marchera très bien (après le message d'erreur (ERR)) et remplira le tableau de 0 supplé-

mentaires.
Alors que:

```
DIM B (5, 5, 3, 5)
RECALL B
```

fonctionnera après le message d'erreur (ERR) mais en brouillant les nombres et:

```
DIM B (5, 5, 1, 1)
RECALL B
```

provoquera le message:

```
?OUT OF MEMORY ERROR
```

car les 6 * 6 * 2 * 2 éléments du tableau B rappelé (RECALL) sont insuffisants pour contenir les 6 * 6 * 6 éléments du tableau A conservé (STORE).

Les commandes STORE et RECALL ne sont effectives que pour les tableaux d'entiers et de "réels". Les chaînes de caractères doivent d'abord être converties en entiers à l'aide de la fonction ASC avant d'être conservées (STORE).

Bien que STORE et RECALL fassent référence à des nombres de variables sans renseignements sur les dimensions, ces commandes ne marchent que pour des tableaux d'entiers et de réels. Le programme suivant:

```
100 A (3) = 45
110 A = 27
120 STORE A
```

conserve sur cassette les éléments du tableau de A (0) à A (10) (souvenez-vous que les tableaux sont dimensionnés à 11 éléments par défaut) et non par la variable A (qui vaut 27 dans le programme).

Aucun message n'indique à l'utilisateur qu'il doit mettre en route son magnétophone sur l'écoute pour rappeler (RECALL) un tableau, et sur enregistrement pour le conserver (STORE) quand ces commandes s'exécutent. Un "bip" sonore indiquera le début et la fin du tableau.

Le programme:

```
300 DIM B (5, 13)
310 B = 4
320 RECALL B
```

lit sur la cassette les 34 éléments (6 * 14) du tableau de B (0, 0) à B (5, 13). La valeur de la variable B n'est pas changée.

Si jamais vous utilisez STORE ou RECALL sans avoir préalablement dimensionné le tableau, ou si vous n'avez pas utilisé l'index dans le programme, le message:

```
?OUT OF MEMORY ERROR (plus de mémoire, erreur) s'affiche.
```

En mode d'exécution immédiate, si vous utilisez STORE et RECALL se référants à un tableau utilisé dans un programme, la ligne de déclaration du programme doit avoir été exécutée avant de conserver ou de rappeler le tableau.

Seul **RESET** interrompt STORE et RECALL.

Si les mots STORE et RECALL sont utilisés comme premiers caractères d'un nom de variable, les deux commandes s'exécutent avant le message: ?SYNTAX ERROR (erreur de syntaxe).

L'instruction:

STORE HOUSE = 5

provoquera l'affichage de:

?OUT OF DATA ERROR (plus de données, erreur) à moins que vous ayez préalablement déclaré un tableau dont le nom commençant par HO. Dans ce cas, B.E.V.F. essaiera de conserver (STORE) le tableau, les "bip" se feront entendre et le message: ?SYNTAX ERROR s'affichera lorsque le B.E.V.F. essaiera d'interpréter le reste de l'instruction (=5).

Pour éviter l'erreur, enfoncez **RESET**.

L'instruction:

RECALLOUS = 234

provoquera de même un: ?OUT OF DATA ERROR à l'affichage, sauf si un tableau dont le nom commence par OU a été préalablement défini.

Dans ce cas, le B.E.V.F. attendra indéfiniment un tableau sur le magnétophone.

Le seul moyen de reprendre le contrôle de l'ordinateur est **RESET**.

Notes

CHAPITRE 6

COMMANDES D'ENTREES/SORTIES

65

```
INPUT          pg seulement
INPUT          [chaîne;] var [{}, var{}
```

Si la chaîne optionnelle est omise, INPUT affiche un point d'interrogation et attend que l'utilisateur entre un nombre (si var est une variable arithmétique) ou des caractères (si var est une variable de chaîne) la valeur de ce nombre ou de cette chaîne est placée dans var.

Quand la chaîne est présente elle s'affiche sur l'écran mais sans point d'interrogation, sans espace ou toute autre forme de ponctuation.

Vous ne pouvez utiliser qu'une chaîne de caractères. Elle doit apparaître immédiatement après INPUT et être terminée par un (;) point-virgule.

Les INPUT numériques devront être des entiers ou des réels, pas d'expressions arithmétiques. Les caractères espaces, +, -, E et la marque décimale font partie du nombre et peuvent être tapés à l'INPUT. Toute concaténation de ces caractères présentée dans une bonne syntaxe (Exemple: + E - est bon alors que + - ne l'est pas).

Quelle que soit la position des espaces, ils sont ignorés. Si dans un INPUT numérique le format n'est pas correct, la question ?REENTER (ré-entrer) s'affiche et l'INPUT recommence.

Si vous utilisez ONERR GOTO, avec un saut (GOTO) qui demande au programme de refaire l'INPUT, la 86ème erreur d'entrée (INPUT) risque d'envoyer le programme dans le mode langage machine. Pour récupérer le programme, faites

CTRL
C

 puis

RETURN

.

On peut éviter ce problème en utilisant RESUME pour revenir à l'instruction INPUT.

De manière équivalente, une réponse assignée à une variable alphanumérique doit être une chaîne de caractères, ou du texte, mais pas une expression de chaîne.

Les espaces précédant le premier caractère sont ignorés. Si la réponse est une chaîne, un guillemet n'importe où dans la chaîne provoquera le message:

```
?REENTER
```

Cependant, dans une chaîne TOUS les caractères, hormis le guillemet,

CTRL
X

,

CTRL
M

, sont acceptés. Cela inclut les deux points et la virgule, les espaces derrière le dernier caractère de la chaîne sont ignorés.

Si la réponse est un littéral, les guillemets sont alors acceptés comme caractères n'importe où dans la chaîne, sauf pour le premier caractère n'étant pas un espace.

Cependant, la virgule, les deux points,

CTRL
X

 et

CTRL
M

 ne sont pas des caractères acceptés

Si l'utilisateur enfonce uniquement

RETURN

 pour un INPUT le message:

```
?REENTER s'affiche et toute l'instruction INPUT se réexécute. Si 

|        |
|--------|
| RETURN |
|--------|

 est enfoncé pour une réponse à un INPUT demandant une chaîne, la réponse est interprétée comme la chaîne nulle et l'exécution du programme continue.
```

Les variables indiquées successivement dans l'instruction INPUT doivent être communiquées

successivement par l'utilisateur. Les variables alphanumériques et numériques peuvent être mélangées, mais chaque réponse devra correspondre au type attendu par l'instruction. Les différentes réponses aux variables doivent être séparées par des virgules ou par des

RETURN

Si l'utilisateur tape des virgules dans une réponse qui ne commence pas par un guillemet, les virgules sont interprétées comme des séparateurs de réponses, même si une seule réponse est attendue.

Si les deux points sont tapés dans une réponse à un INPUT qui ne commence pas par un guillemet, tous les caractères après les deux points sont ignorés. Après les deux points, les virgules sont aussi ignorées, donc le début d'une autre réponse devra être signalé par un **RETURN**.

Si un **RETURN** est pressé avant que toutes les variables arithmétiques aient été introduites, deux points d'interrogation sont affichés pour indiquer qu'une réponse supplémentaire est nécessaire. Quand **RETURN** est pressé et que la réponse contient plus de champs de réponses (séparés par une virgule) ou si les deux points sont présents dans la dernière des réponses, le message:

?EXTRA IGNORED (suppléments ignorés) s'affiche et l'exécution continue.

Si les deux points ou une virgule sont les premiers caractères d'une réponse à un INPUT la réponse est évaluée à \emptyset ou à la chaîne nulle.

Notez bien que dans l'instruction INPUT la chaîne de caractères optionnelle doit être suivie d'un point-virgule, mais les variables doivent être séparées par des virgules.

CTRL
C peut interrompre une instruction INPUT mais seulement si c'est le premier caractère

tapé. Le programme s'arrête quand **RETURN** est enfoncé. Un essai de continuation du programme (**CONT**) provoquera un: ?SYNTAX ERROR (erreur de syntaxe) qui s'affiche.

CTRL
C est considéré comme tout autre caractère s'il n'est pas tapé comme premier caractère

de la réponse.

Utiliser INPUT en mode d'exécution immédiat provoquera l'affichage de l'erreur:

?ILLEGAL DIRECT ERROR (erreur de mode immédiat).



Si une erreur se produit pendant l'introduction d'une variable, l'INPUT recommence avec la première variable.

GET pg seulement
GET var

Demande un caractère du clavier sans l'afficher sur l'écran et sans que **RETURN** ait besoin d'être pressé. GET avec une variable de chaîne peut parfois avoir des comportements curieux.

CTRL
C renvoie un espace.

Utiliser GET en tapant la flèche à gauche ou **CTRL**
H pour renvoyer l'espace sur l'écran.

CTRL
C est considéré comme tout autre caractère et n'interrompt pas l'exécution du programme.

L'instruction GET n'a pas été conçue pour être utilisée avec des variables arithmétiques, vous pouvez utiliser:

GET vara, en notant bien que:



utiliser GET en introduisant une virgule ou les deux points provoquera l'affichage de: ?EXTRA IGNORED (supplément ignoré), suivi de l'affichage d'un zéro considéré comme la valeur tapée.

Les signes plus ou moins,

CTRL
C

, E, l'espace et le point renvoient zéro comme valeur introduite.

Introduire une valeur non numérique provoque le message:

?SYNTAX ERROR (erreur de syntaxe) à l'affichage.

En utilisant ON ERROR GOTO...RESUME, deux erreurs consécutives sur un GET feront que l'ordinateur bouclera dans son système jusqu'à ce que

RESET

 soit utilisé.

Si vous utilisez GOTO au lieu de RESUME tout va bien jusqu'à ce que vous commétez la 43ième erreur de GET (de tout type), le programme s'arrête alors en mode langage machine. Pour le récupérer, faites

CTRL
C

RETURN

.

A cause de ces limitations il est conseillé de n'utiliser que:

GET varc et de convertir la chaîne résultante en nombre en utilisant la fonction VAL.

```
DATA      pg seulement
DATA      [[littéral|chaîne|réel|entier] {},[littéral|chaîne|réel|entier]{}]
```

Cette instruction crée une liste d'éléments qui peuvent être lus par l'instruction READ. En respectant l'ordre des numéros de ligne, chaque instruction DATA ajoute ses éléments à la liste déjà existante des autres éléments créés par les anciennes instructions DATA que vous avez déjà tapées.

L'instruction DATA ne doit pas forcément précéder l'instruction READ dans un programme, les instructions DATA peuvent s'implanter n'importe où dans le programme.

Les données (DATA) qui sont lues (READ) dans des variables arithmétiques suivent les mêmes règles de format en général que pour les entrées de l'introduction INPUT pour les variables arithmétiques.

Cependant les deux points ne peuvent être introduits comme un caractère dans un élément DATA. Si

CTRL
C

 est un élément de données (DATA) il n'arrête pas le programme, même si

c'est le premier caractère d'un élément. Hormis cette exception, les éléments DATA qui sont lus (READ) par des variables alphanumériques suivent les mêmes règles que les réponses aux INPUT des variables alphanumériques.

On peut utiliser des chaînes ou des littéraux, ou les deux. Les espaces avant le premier caractère et après le dernier sont ignorés.

Tout guillemet apparaissant dans une chaîne provoque le message: ?SYNTAX ERROR (erreur de syntaxe), mais tout autre caractère sera accepté dans une chaîne (même les deux points et la virgule) sauf

CTRL
X

 et

CTRL
M

.

Si un élément est un littéral, le guillemet est un caractère accepté sauf comme premier caractère différent de l'espace de la chaîne. Les deux points, la virgule,

CTRL
X

 et

CTRL
M

 ne sont pas acceptés. Référez-vous à INPUT pour plus de détails.

Les éléments de données (DATA) peuvent être un mélange de réels, d'entiers, de chaînes et de littéraux.

Si un READ essaye d'assigner une chaîne ou un littéral à une variable arithmétique, le message: ?SYNTAX ERROR s'affiche pour la ligne DATA appropriée.

Si une liste de DATA contient un élément vide, un zéro (numérique) ou la chaîne vide est assignée à la variable du READ.

Un élément est vide si:

- 1 - il n'y a pas de caractère différent de l'espace entre DATA et

RETURN

.
- 2 - il n'y a pas de caractère différent de l'espace entre deux virgules
- 3 - la virgule est le premier caractère différent de l'espace dans le DATA
- 4 - la virgule est le dernier caractère différent de l'espace dans le DATA

Donc si vous écrivez:

```
100 DATA,,
```

une instruction READ pourra y lire trois éléments qui seront, soit des zéros, soit des chaînes nulles, cela dépendant du type de variable utilisée.

Utilisé en mode immédiat, DATA ne provoque pas de: ?SYNTAX ERROR mais il est inutilisable par un READ.

```
READ      imm et pg  
READ      var [ }, var{ ]
```

Quand la première instruction READ s'exécute dans un programme, la première des variables prend la valeur du premier élément du premier DATA (la liste des données (DATA) consiste en tous les éléments de toutes les instructions DATA) rencontré dans le programme. S'il y a une seconde variable, elle prend la valeur du deuxième élément de la liste des données (DATA), et ainsi de suite.

Quand l'instruction READ a fini de s'exécuter, elle conserve un POINTEUR DE LISTE DE DONNEES sur l'élément suivant le dernier élément lu par READ.

La prochaine instruction READ (s'il y en a une) commencera sa lecture à partir de l'élément qu'indique le pointeur de liste.

Seuls RUN ou RESTORE placent le pointeur sur le premier élément de la liste de données (DATA).

Essayer de lire plus d'éléments que la liste des DATA en contient provoquera le message:

```
?OUT OF DATA ERRO IN numligne (plus de données en ligne) où numligne est le numéro de ligne de l'instruction READ où l'erreur s'est produite.
```

En mode immédiat, vous ne pouvez lire que des éléments qui existent dans les DATA d'un programme mémorisé. Les éléments peuvent être lus (READ) même si le programme n'a pas encore été exécuté. S'il n'y a pas d'instruction DATA dans le programme, le message:

```
?OUT OF DATA ERROR s'affiche.
```

Exécuter des instructions en mode immédiat ne place pas le pointeur sur le premier élément de la liste.

Les données supplémentaires qui ne sont jamais lues sont acceptées.

```
RESTORE    imm et pg
RESTORE
```

Cette instruction sans parenthèses ou options place le pointeur d'éléments de liste de données (voir DATA et READ) sur le premier élément de la liste.

```
PRINT      imm et pg
PRINT      [{expr}{[,|;[{expr}]}][,|;]
PRINT      ;{
PRINT      },} },{
```

Le point d'interrogation (?) peut être utilisé comme abréviation de l'instruction PRINT, le mot PRINT réapparaîtra au LISTING.


Sans option ni paramètres le PRINT provoque un saut de ligne sur l'écran. Avec des options PRINT affiche sur l'écran les valeurs de la liste qui l'accompagne. Si, ni une virgule, ni un point-virgule ne terminent la liste, un saut de ligne est effectué après l'affichage du dernier article (texte, chaîne, variables...) de la liste.

Si un article de la liste est suivi d'une virgule, l'affichage du prochain article se fera en première colonne du prochain champ de tabulation de l'écran disponible.

Le premier champ de tabulation comprend les 16 colonnes d'affichage les plus à gauche de la fenêtre de texte (position 1 à 16). Le second champ s'étend sur les 16 colonnes d'affichage suivantes (de 17 à 32), la disponibilité d'impression n'est valable sur ce second champ pour autant que rien ne soit imprimé en position 16.

Le 3ème champ de tabulation est constitué par les 8 colonnes d'affichage restantes (33 à 40) et est disponible à l'impression pour autant que rien ne soit imprimé de la position 24 (inclusive) à 32 (inclusive).

La taille de la fenêtre de texte peut être modifiée en utilisant les commandes POKE (voir annexe J).

 Le 3ème champ de tabulation ne fonctionnera pas correctement si la largeur de la fenêtre de texte est inférieure à 33 colonnes. Le premier caractère de ce champ risque d'être affiché (PRINT) à l'extérieur de la fenêtre. HTAB peut aussi faire que PRINT se fasse à l'extérieur de la fenêtre.

Un article de la liste suivi par un point-virgule provoquera l'affichage de PRINT de l'article suivant collé au premier article sans espace.

Des articles de la liste peuvent être affichés collés sans aucun problème en n'utilisant pas les ; et les , s'il n'y a pas de problèmes d'interprétation pour le B.E.V.F.

Voici un exemple illustrant cela:

```
A = 1 : D = 2 : C = 3 : C (4) = 5 : C5 = 7
PRINT 1/3 (2*4) 51,
.333 333 333 851
```



```
PRINT 1 (A) 2 (B) 3C (4) C5
11 22 357
PRINT 3.4.5.6.,
3.4.5.6Ø
PRINT A. "B." C.4
1Ø B.3.4
```

PRINT est une instruction très puissante pour faire afficher ce que vous désirez. S'il ne peut interpréter un point comme la virgule décimale, il le considère comme le chiffre Ø, comme l'illustrent les exemples précédents.

PRINT suivi d'une liste de points-virgules n'en fait pas plus que PRINT tout court, mais c'est parfaitement admis.

PRINT suivi d'une liste de virgules espace l'affichage d'un champ de tabulation par virgule, jusqu'à ce que la limite de 239 caractères dans une instruction, soit atteinte.

```
PRINT A$ + B$
affiche un:
```

?STRING TOO LONG ERROR (chaîne trop longue, erreur) si la longueur de la chaîne concaténée est plus grande que 255 caractères. Vous pouvez alors utiliser une APPARENTE concaténation avec:

```
PRINTA$ B$
sans vous soucier des longueurs.
```

```
IN#      imm et pg
IN#      extra
```

Sélectionne une entrée du connecteur d'entrée/sortie numéro \extra\ . Utilisé pour spécifier quel périphérique sera appelé. Le périphérique doit être installé sur les connecteurs numérotés de 1 à 7, indiqués par \extra\ .

IN # Ø indique que la prochaine entrée se fera à nouveau du clavier et non du périphérique. On ne peut pas utiliser en B.E.V.F. le slot Ø pour y connecter un périphérique. S'il n'y a pas de périphérique sur le connecteur extra le système se met à boucler. Pour récupérer le contrôle de l'ordinateur faire **RESET** **CTRL** **RETURN** .
C

Si \extra\ est inférieur à Ø ou supérieur à 255, le message :
?ILLEGAL QUANTITY ERROR (quantité illégale, erreur) s'affiche.

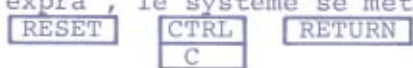


Si \extra\ est compris entre 8 et 255, vous risquez une altération imprévisible du B.E.V.F. Pour les transferts de sortie voir PR#.

```
PR#      imm et pg
PR#      extra
```

Sélectionne une sortie sur le connecteur d'entrée/sortie spécifié par \extra\ , \extra\ doit être compris entre 1 et 7.

PR # Ø renvoie la sortie sur la télévision, et non pas sur le connecteur Ø.

S'il n'y a pas de périphérique sur le connecteur `expra`, le système se met à boucler. Pour récupérer le contrôle de l'ordinateur, faire `RESET` `CTRL` `RETURN`.


Si `\expra\` est inférieur à 0 ou supérieur à 255, le message:
 ?ILLEGAL QUANTITY ERROR (quantité illégale, erreur) s'affiche.



Si `\expra\` est compris entre 8 et 255, vous risquez une altération imprévisible du B.E.V.F.

Pour les transferts d'entrée voir IN#.

```
LET          imm et pg
[LET]       vara [dimension] = expra
[LET]       varc [dimension] = exprc
```

Au nom de la variable à gauche du signe égal est assignée la valeur de la chaîne ou de l'expression arithmétique sur la droite. Le LET est optionnel.

```
LET A = 2
```

et

```
A = 2
```

sont deux instruction identiques.

Le message:

?TYPE MISMATCH ERROR (erreur de type) s'affiche si vous essayez de donner:

- a - un nom de variable alphanumérique à une expression arithmétique
- b - un nom de variable alphanumérique à un littéral
- c - un nom de variable numérique à une expression de chaîne de caractères.

Si vous essayez de donner un nom de variable arithmétique à un littéral, le B.E.V.F. essaie d'interpréter le littéral comme une expression arithmétique.

```
DEF          pg
FN           imm et pg
DEF         FN nom (vara réelle) = expra 1
FN          nom (expra 2)
```

Permet à l'utilisateur de définir des fonctions dans un programme.

La fonction FN nom est d'abord définie avec DEF.

Une fois qu'une ligne de programme définissant la fonction a été exécutée, la fonction peut être utilisée sous la forme FN nom (argument) ou l'argument `\expra 2\` peut être une expression arithmétique.

La Définition d'`\expra 1\` doit tenir sur une seule ligne de programme; la fonction définie FN nom peut être utilisée partout où le B.E.V.F. accepte les fonctions arithmétiques.

Il est possible de reDéfinir les fonctions au cours d'un programme. Les règles concernant les variables arithmétiques doivent être conservées pour les fonctions.

En particulier les deux premiers caractères définissant un nom doivent être uniques.

Quand les lignes:

```
10 DEF FN ABC (I) = COS (I)
20 DEF FN ABT (I) = SIN (I) * TAN (I)
```

sont exécutées, B.E.V.F. reconnaît en ligne 10 la fonction d'une définition FN AB et redéfinit la même fonction FN AB en ligne 20.

Dans la DEFinition d'une fonction, vara réel est une variable "prête-nom". Quand la fonction FN non définie est utilisée, plus tard: elle est appelée avec l'argument \expr 2\. Cet argument se substitue à vara réelle à tout endroit où vara réelle apparaît dans la définition \expr 1\. \expr 1\ peut contenir autant de variables que désiré, mais bien sûr, seulement une au plus correspond à la variable "prête-nom", et donc à la variable considérée comme argument.

Il n'est pas obligé que vara réelle apparaisse dans \expr 1\, mais alors quand la fonction est utilisée plus tard dans le programme, l'argument de la fonction n'entre pas dans le calcul de expr 1. Mais, même dans ce dernier cas, la valeur de la fonction est calculée, \expr 1\ doit donc être quelque chose d'autorisé.

Exemple:

```
100 DEF FN A (W) = 2 * W + W
110 PRINT FN A (23)
120 DEF FN B (X) = 4 + 3
130 G = FN B (23)
140 PRINT G
150 DEF FN A (Y) = FN B (Z) + Y
160 PRINT FN A (G)
```

```
RUN
69      [ FN A (23) = 2 * 23 + 23 ]
7       [ FN B (N'IMPORTE QUOI) = 7 ]
14      [ NOUVELLE FN A (7) = 7 + 7 ]
```

Si dans un programme DEF FN nom n'est pas exécuté avant d'utiliser FN nom, le message:

?UNDEF'D FUNCTION ERROR (fonction non définie, erreur) s'affiche sur l'écran. Les fonctions utilisateurs de chaîne de caractères ne sont pas autorisées. Les fonctions définies en utilisant un nom entier (%) ou une vara réelle entière ne sont pas autorisées.

Quand une nouvelle fonction est DEFinie, 6 octets sont utilisés pour stocker en mémoire le pointeur de cette définition.

Notes



CHAPITRE 7

COMMANDES RELATIVES AUX BRANCHEMENTS

```
GOTO      imm et pg
GOTO      numligne
```

Branchement inconditionnel à la ligne de numéro numligne. Si le numéro n'existe pas dans le programme, ou s'il est absent dans l'instruction GOTO, le message:

?UNDEF'D STATEMENT ERROR IN numligne (instruction non définie en ligne numligne) s'affiche où le numligne est le numéro de la ligne indiqué dans l'instruction GOTO.

```
IF        imm et pg
IF        expr THEN instruction [{}: instruction{}
IF        expra [THEN] GOTO numligne
IF        expr THEN [GOTO] numligne
```

Si `\expr\` est une expression arithmétique dont la valeur est différente de zéro (et dont la valeur absolue est plus grande qu'environ $2.93873E -39$), `\expr\` est considérée comme vraie et toute(s) instruction(s) suivant THEN est exécutée.

Si `\expr\` est une expression arithmétique dont la valeur est égale à zéro (ou dont la valeur absolue est plus petite que $2.93873E -39$), toute(s) instruction(s) suivant le THEN est ignorée et le programme continue à la ligne numérotée suivante.

Si une instruction IF s'exécute en mode immédiat et si `\expr\` vaut zéro, le B.E.V.F. ignorera toute autre instruction tapée en mode immédiat.

Si `expr` est une expression contenant des expressions de chaînes de caractères et des opérateurs logiques de chaîne, `\expr\` s'évalue en comparant le rang alphabétique des expressions, conformément aux normes du code ASCII concernant les caractères (Cf. annexe K).

Les instructions de la forme:

```
IF expr THEN
```

sont valables: pas de message d'erreur.

Un THEN sans IF correspondant et un IF sans THEN correspondant provoquera le message:

?SYNTAX ERROR (erreur de syntaxe) à l'affichage.

Le B.E.V.F. n'est pas conçu pour utiliser des instructions IF avec comme `\expr\` des chaînes de caractères, mais les variables de chaînes et les chaînes peuvent être utilisées comme `\expr\` sous les contraintes suivantes:

si `\expr\` est une expression de chaîne quelconque, alors `\expr\` est non nulle, même si `\expr\` est une variable de chaîne qui n'a pas été assignée ou qui a été assignée à "Ø" ou à la chaîne nulle " ".

Cependant le littéral nul:

```
IF "" THEN...
```

est évalué à zéro.



Si IF chaîne THEN... est exécutée plus de deux ou trois fois dans un programme donné, le message: ?FORMULA TOO COMPLEX ERROR (formule trop complexe, erreur) s'affiche.



Si \expr\ est une variable de chaîne et que dans les instructions précédentes la chaîne nulle a été assignée à N'IMPORTE quelle variable de chaîne, alors \expr\ est évalué à zéro.

Par exemple le programme:

```
120 IF A$ THEN PRINT "A$"
130 IF B$ THEN PRINT "B$"
140 IF X$ THEN PRINT "X$"
```

en l'exécutant (RUN) il s'affiche:

```
A$
B$
X$
```

Car les chaînes A\$, B\$ et X\$ ne sont pas évaluées à zéro. Cependant en ajoutant la ligne:

```
100 Q$ = ""
```

les 3 chaînes sont évaluées à zéro et il n'y a aucun affichage. Supprimer la ligne 100 ou rajouter par exemple la ligne 110 telle que:

```
110 F = 3
```

provoque une évaluation des 3 chaînes à une valeur différente de zéro et l'affichage se refait.

La lettre A immédiatement avant THEN pose des problèmes d'interprétation.



IF BETA THEN est interprété comme:

```
IF BET AT HEN 130
```

et génère un message:

```
?SYNTAX ERROR (erreur de syntaxe) sur l'écran.
```

Ces trois formes sont équivalentes:

```
IF A = { THEN 160
```

```
IF A = { THEN GOTO 160
```

```
IF A = { GOTO 160
```

```
FOR      imm et pg
```

```
FOR      vara réelle = expr 1 TO expr 2 [STEP expr 3]
```

\vara\ prend la valeur d'\expr 1\, et les instructions après le FOR sont exécutées jusqu'à ce qu'une instruction:

NEXT vara

soit rencontrée par le programme, ou \vara\ est le même que \vara\ rencontré dans l'instruction FOR.

Puis \vara\ s'incrémente de \expra 3\ (si \expra 3\ n'apparaît pas dans l'instruction FOR l'ordinateur considère l'incrément de 1).

La nouvelle valeur de \vara\ est comparée à \expra 2\ et si \vara\ > \expra 2\ l'exécution du programme continue avec les instructions suivant le NEXT.

Si \vara\ <= \expra 2\, l'exécution reprend à l'instruction immédiatement après le FOR.

Les expressions arithmétiques (expra) formant les paramètres de la boucle FOR peuvent être des variables réelles, des variables entières, des réels ou des entiers.

Cependant, \vara réelle\ DOIT être une variable réelle. Le fait d'utiliser une variable entière à la place de vara réelle, provoquera le message:

?SYNTAX ERROR (erreur de syntaxe) sur l'écran.

Comme \vara réelle\ s'incrémente et est comparée à \expra 2\ seulement à l'exécution de l'instruction NEXT, la partie du programme entre FOR et NEXT est exécutée au moins une fois.

Plusieurs boucles FOR...NEXT ne peuvent s'entrecroiser (elles doivent s'imbriquer). Si elles le sont, le message:

?NEXT WITHOUT FOR ERROR (NEXT sans FOR, erreur) s'affiche sur l'écran.

Si vous imbriquez plus de 10 boucles FOR...NEXT, le message:

?OUT OF MEMORY ERROR (plus de mémoire, erreur) s'affichera.

Pour exécuter une boucle FOR...NEXT en mode immédiat, les instructions FOR et NEXT doivent être écrites sur la même ligne (une ligne peut avoir jusqu'à 239 caractères de long) en utilisant les séparateurs (:)



Si la lettre A est tapée immédiatement avant le TO, n'écrivez pas d'espace entre le T et le O

FOR I = BETA TO 56 est bon, mais:

FOR I = BETA T O 56 est interprété comme:

FOR I = BET AT O56 et provoque l'affichage du message:

?SYNTAX ERROR sur l'écran à l'exécution.

Chaque boucle FOR...NEXT utilise 16 octets en mémoire.

```
NEXT      imm et pg
NEXT      [vara réelle]
NEXT      vara réelle [}, vara réelle{]
```

Ferme une boucle FOR...NEXT. Quand le programme rencontre un NEXT, il l'ignore ou se branche à l'instruction suivant immédiatement le FOR. Cela dépend des conditions expliquées à l'instruction FOR.

Si plusieurs vara réelles sont spécifiées, elles doivent être utilisées dans le bon ordre pour éviter que les boucles ne s'entrecroisent et provoquent le message:

?NEXT WITHOUT FOR ERROR (NEXT sans FOR, erreur) sur l'écran.

Une instruction NEXT utilisée sans nom de variable se réfère à la plus récente instruction FOR en cours d'exécution. Si NEXT sans paramètres ne se réfère pas à une instruction le FOR le message:

?NEXT WITHOUT FOR ERROR s'affiche.

NEXT sans vara réelle s'exécute plus rapidement que NEXT avec vara réelle

En mode immédiat, les instructions FOR et NEXT doivent être placées sur la même ligne à l'aide des séparateurs (:). Si une instruction FOR d'un programme est toujours sans effet, un NEXT en mode immédiat provoque la continuation de l'exécution du programme. Cependant, si la boucle a été ouverte par un FOR en mode immédiat, un NEXT, toujours en mode immédiat, mais sur une ligne différente, provoquera le message:

?SYNTAX ERROR (erreur de syntaxe)
sauf s'il n'y a pas de lignes intermédiaires et que le NEXT est sans paramètres:

```
]FOR I = 1 TO 5 : PRINT I
1
]NEXT
2
]NEXT I
?SYNTAX ERROR IN XXXX (ne pas tenir compte de XXXX)
```

GOSUB imm et pg
GOSUB numligne

Le programme, pour exécuter un sous-programme, saute à la ligne indiquée par numligne. Quand l'instruction RETURN est rencontrée, le programme saute à l'instruction suivant immédiatement le GOSUB exécuté le plus récemment.

Chaque fois qu'un GOSUB s'exécute, l'adresse de retour (adresse de l'instruction suivant le GOSUB) est conservée au moyen d'une "pile" (structure "LIPO" dernière entrée, première sortie) et le programme peut alors retrouver ultérieurement son retour au programme principal.

A chaque fois qu'un RETURN ou un POP s'exécute, l'adresse du sommet de la pile est enlevée. Si le numligne indiqué n'existe pas dans le programme, le message:

?UNDEF'D STATEMENT ERROR IN numligne (instruction non définie en numligne, erreur) s'affiche, ou numligne indique la ligne de programme contenant l'instruction GOSUB.

La partie IN numligne du message d'erreur n'apparaît pas avec les GOSUB utilisés en mode immédiat. Si vous indiquez plus de 25 niveaux de sous-programmes (GOSUB) le message:

?OUT OF MEMORY ERROR (plus de mémoire, erreur) s'affiche sur l'écran.

Les sous-programmes (GOSUB) ne doivent pas s'entrecroiser. Chaque GOSUB en cours d'exécution (qui n'a pas encore rencontré de RETURN) utilise 6 octets de mémoire.

RETURN imm et pg
RETURN

Il n'y a pas de parenthèses ou options pour cette instruction. RETURN exécute un saut à l'instruction suivant le dernier sous-programme appelé (GOSUB).

L'adresse de l'instruction suivant le GOSUB est conservée dans le fichier des RETURN.
(Voir GOSUB et POP).

Si le programme rencontre plus d'instructions RETURN que de GOSUB le message:
?RETURN WITHOUT GOSUB ERROR (RETURN sans GOSUB, erreur) s'affiche car il n'y a plus
d'adresses de RETOUR dans le fichier.

```
POP      imm et pg
POP
```

Il n'y a pas de paramètres ou options pour cette instruction. Le POP a le même effet que
RETURN mais il n'y a pas de branchement. L'exécution continue à la ligne suivante.
La prochaine instruction RETURN rencontrée ne provoque pas de retour à l'instruction sui-
vant le dernier sous-programme appelé (GOSUB) mais à l'instruction suivant L'AVANT dernier
sous-programme appelé. POP permet de sauter un niveau de retour de sous-programme.
Si une instruction POP est exécutée avant l'appel d'un sous-programme (GOSUB),
?RETURN WITHOUT GOSUB ERROR (RETURN sans GOSUB, erreur) s'affiche sur l'écran car il n'y
a pas d'adresse RETURN sur la file.

```
ON...GOTO  pg
ON...GOSUB pg
ON expra GOTO numligne {[, numligne]}
ON expra GOSUB numligne {[, numligne]}
```

ON...GOTO saute à la ligne de programme spécifié par le \expra\ième article de la liste
des numligne écrite après GOTO.

ON...GOSUB fonctionne similairement hormis qu'il s'adresse aux sous-programmes (GOSUB) et
non aux branchements inconditionnels (GOTO).

Si \expra\ vaut 0 ou est supérieur au nombre de numéros de lignes indiqués dans l'instruc-
tion, l'exécution continue à l'instruction suivante.
\expra\ doit être compris entre 0 et 255 pour éviter le message:
?ILLEGAL QUANTITY ERROR (quantité illégale, erreur).

```
ONERR GOTO pg seulement
ONERR GOTO numligne
```

Si une erreur se produit, l'instruction ONERR GOTO permet d'éviter l'affichage du message
d'erreur et l'arrêt d'exécution du programme. L'instruction positionne un indicateur qui
provoque un branchement inconditionnel (si une erreur se produit plus tard dans le program-
me) à la ligne indiquée par numligne. POKE 216,0 réinitialise l'indicateur de détection
d'erreur et le message d'erreur s'affichera alors normalement.

Quand une erreur se produit dans un programme le code indiquant le type d'erreur se trouve
à la case mémoire d'adresse décimale 222.

Pour voir de quelle erreur il s'agit: PRINT PEEK (222).

<u>CODE</u>	<u>TYPE D'ERREUR</u>
∅	NEXT sans FOR
16	syntaxe
22	RETURN sans GOSUB
42	plus de données (DATA)
53	quantité illégale
69	dépassement de capacité
77	plus de mémoire
90	instruction non définie
107	index mauvais
120	tableau redimensionné
133	division par zéro
163	erreur de type
176	chaîne trop longue
191	formule trop complexe
224	fonction non définie
254	mauvaise réponse à une instruction INPUT
255	essai d'interruption par un CTRL C

➔ On doit faire attention en manipulant les erreurs pour éviter l'affichage des messages et l'arrêt du programme, quand l'erreur se produit dans une boucle FOR...NEXT ou dans un sous-programme (GOSUB...RETURN), en effet les pointeurs ou le fichier RETURN sont perturbés.

Un branchement avec ONERR GOTO numligne sur un NEXT ou un RETURN provoqueront l'erreur:
?NEXT WITHOUT FOR ERROR (NEXT sans FOR, erreur)

ou bien

?RETURN WITHOUT GOSUB ERROR (RETURN sans GOSUB, erreur)

➔ Quand on utilise ONERR GOTO et que RETURN est utilisé dans le programme de manipulation des erreurs qui commence à la ligne indiquée par numligne. Si l'erreur se produit plus de deux fois de suite sur une instruction GET le programme décroche et se met à boucler. Pour récupérer le contrôle de l'ITT 2020 faire

RESET CTRL
C RETURN

Si GOTO termine le programme de manipulation des erreurs, alors tout se passe bien.

➔ Utilisé en mode dépiage (TRACE) ou dans un programme contenant une instruction PRINT, ONERR GOTO peut provoquer un saut dans le langage machine après 43 détections d'erreurs.

Si des erreurs sont détectées à une instruction INPUT et que vous utilisez un GOTO dans le programme de manipulation des erreurs, alors après la 87ème détection d'erreur, le programme sautera en langage machine. RESET CTRL
C et RETURN vous feront récupérer votre

programme. Pour éviter ce saut, utiliser RESUME dans le programme de manipulation des erreurs.

Si les problèmes ci-dessus vous tracassent, appelez par un CALL le programme en langage machine ci-dessous dans votre programme de manipulation des erreurs et les problèmes seront réglés.

En moniteur entrez les données hexadécimales suivantes:

68 A8 68 A6 DF 9A 48 98 48 60

ou entrez en B.E.V.F. les données décimales suivantes:

104 168 104 166 223 154 72 152 72 96

Vous auriez pu, en B.E.V.F., introduire par exemple les données par des POKE à partir de l'adresse 768; puis utiliser CALL 768 dans votre programme de manipulation des erreurs.

RESUME pg
RESUME

Pas de paramètres. Utilisée à la fin d'un programme de manipulation des erreurs, cette instruction provoque une reprise de programme au début de l'instruction qui provoque l'erreur.

Si RESUME s'exécute avant qu'une erreur arrive, le message:

?SYNTAX ERROR IN 65 278 (erreur de syntaxe en 65 278) s'affiche, ou bien d'autres événements bizarres risquent d'arriver.

En général, votre programme s'arrêtera ou se mettra à boucler indéfiniment.

Si une erreur se produit dans le programme de manipulation d'erreurs, RESUME fera que le programme se mettra à boucler indéfiniment. Pour récupérer le contrôle de l'ordinateur:

RESET CTRL RETURN.
 C

En mode immédiat RESUME peut faire boucler le système, ou afficher un ?SYNTAX ERROR ou encore exécuter un programme existant ou MEME DETRUIT!

Notes



CHAPITRE 8

GRAPHISME

(et leviers de commande)

(Ces "leviers" sont fournis seulement en option)

TEXT imm et pg
TEXT

Sans paramètres. Sélectionne le mode texte (caractères) de l'ITT 2020 (40 caractères par ligne, 24 lignes).

Permet de sortir des modes graphiques couleur haute et large résolution. Le caractère de reconnaissance du B.E.V.F. et le curseur sont placés sur la dernière ligne de l'écran. Exécuté en mode texte, TEXT est similaire à VTAB 24.

Une instruction telle que:

```
175 TEXT ILE = 127
```

exécute l'instruction correspondant au mot réservé TEXT avant d'indiquer un:

```
?SYNTAX ERROR (erreur de syntaxe) sur l'écran.
```

Si la fenêtre de texte a été modifiée par l'utilisateur (voir annexe J), TEXT la réinitialise à tout l'écran.

GR imm et pg
GR

Pas de paramètres. Cette commande sélectionne le mode graphique couleur large résolution, laissant quatre lignes de texte en bas de l'écran. L'écran est nettoyé et le curseur déplacé dans la fenêtre de texte. Pour convertir l'écran en tout graphique, la commande POKE -16302,0 ou POKE 49234,0 transforme les 4 lignes de texte en zone graphique. Si vous refaites GR, vous récupérez alors le mode mixte texte-graphique.

Après une commande GR, la couleur (COLOR) s'initialise à 0.



Si le mot réservé GR est utilisé comme premiers caractères d'un nom de variable, GR sera exécuté avant l'affichage du message:

```
?SYNTAX ERROR (erreur de syntaxe) sur le bas de l'écran.
```



Utilisé après un HGR, GR fonctionne mais utilisé après HGR2, GR nettoie sa partie mémoire mais vous laisse la page 2 de la large résolution ou du texte sur l'écran. Pour retrouver l'affichage normal, utilisez TEXT. En programme utilisez TEXT avant de passer de HGR2 en GR.

COLOR imm et pg
COLOR = expra

Cette instruction définit la couleur des petits rectangles en mode graphique large résolution. Si \expra\ est un nombre réel, il est converti en nombre entier. \expra\ doit être compris entre 0 et 255 et l'ordinateur en prend la valeur modulo 16.

Les couleurs sont annoncées aux nombres suivants:

- 0 : noir
- 1 : bleu outremer
- 2 : vert bouteille
- 3 : bleu océan
- 4 : rouge foncé
- 5 : violet
- 6 : ocre
- 7 : mauve
- 8 : marron
- 9 : bleu clair
- 10 : vert pomme
- 11 : turquoise
- 12 : rouge clair
- 13 : vieux rose
- 14 : jaune
- 15 : blanc

L'instruction GR fixe la couleur (COLOR) à 0.
L'instruction SCRN donne la valeur de la couleur (COLOR) pour un point donné.

Utilisé en mode TEXT, l'instruction COLOR détermine le caractère alphanumérique qui sera dessiné (PLOT).

En mode graphique haute résolution, COLOR est sans effet.

PLOT imm et pg
PLOT expra 1, expra 2

En mode graphique large résolution cette instruction dessine un rectangle aux points de coordonnées x = \expra 1\, y = \expra 2\
La couleur du rectangle est déterminée par la dernière instruction COLOR exécutée. (COLOR prend par défaut la valeur 0).

\expra 1\ doit être compris entre 0 et 39, et \expra 2\ entre 0 et 47, sinon on obtient le message:

?ILLEGAL QUANTITY ERROR (quantité illégale, erreur).

Si on utilise PLOT en mode TEXT, ou en mode mixte (Graphisme + 4 lignes de texte) entre 40 et 47, on obtient l'affichage d'un caractère à la place d'un rectangle de couleur. (Un caractère occupe la place de deux rectangles sur la même colonne).

La commande n'a aucun effet visible quand elle est utilisée en mode HGR2, même si elle est précédée de l'instruction GR, du fait que l'écran visualise la page 2 haute résolution et que la commande affecte la page 1 du graphisme large résolution.

L'origine (0, 0) pour tout graphisme est le coin supérieur gauche de l'écran.

```
HLIN      imm et pg
HLIN      expra 1, expra 2 AT expra 3
```

Utilisable en mode graphique large résolution (GR) HLIN trace une ligne horizontale de (\expra 1\, \expra 3\) à (\expra 2\, \expra 3\). La couleur (COLOR) étant déterminée par la dernière instruction COLOR exécutée.

\expra 1\ et \expra 2\ doivent être compris entre 0 et 39.

\expra 3\ doit être compris entre 0 et 47, sinon le message d'erreur:

?ILLEGAL QUANTITY ERROR (quantité illégale, erreur) s'affichera sur l'écran.

\expra 1\ peut être plus grand, égal ou plus petit que \expra 2\.

Si HLIN est utilisé en mode texte (TEXT), ou en mode mixte (Graphisme + 4 lignes de texte) entre 40 et 47. Alors une ligne de caractères se trouvera là où la ligne graphique se serait dessinée. (Un caractère occupe la place de deux rectangles sur la même colonne).

Le H de la commande signifie Horizontal et non haute résolution

A l'exception de HLIN et HTAB, les commandes ayant préfixe "H" se rapportent à une commande haute résolution.

```
VLIN      imm et pg
VLIN      expra 1, expra 2 AT expra 3
```

Trace, en mode graphique large résolution, une ligne verticale de (\expra 1\, \expra 3\) à (\expra 2\, \expra 3\).

La couleur (COLOR) est déterminée par la dernière instruction COLOR exécutée.

\expra 1\ et \expra 2\ doivent être compris entre 0 et 47, \expra 3\ entre 0 et 39 ou le message:

?ILLEGAL QUANTITY ERROR (quantité illégale, erreur) s'affiche.

\expra 1\ peut être plus grand, égal ou plus petit que \expra 2\.

Si l'ordinateur est en mode texte (TEXT) ou en mode mixte (Graphisme + 4 lignes de texte) entre 40 et 47, la portion de ligne dans la zone de texte apparaîtra comme une ligne de caractères alphanumériques, placée là où des caractères graphiques auraient dû apparaître.

La commande n'a pas d'effet visible en mode graphique haute résolution.

```
SCRN      imm et pg
SCRN      (expra 1, expra 2)
```

En mode graphique large résolution, renvoie le code de la couleur du rectangle de coordonnées x = \expra 1\, y = \expra 2\.



Un rectangle dessiné en large résolution est compris entre 0 et 39 pour les x et 0 et 47 pour les y. Cependant la fonction SCRN est utilisée avec x (\expra 1\) entre 40 et 47, le nombre renvoyé donne la couleur (COLOR) du point de coordonnées $x = \text{\expra 1} - 40$ et $y = \text{\expra 2} + 16$.

Si $\text{\expra 2} + 16$ est compris entre 40 et 47 et que le mode Graphique mixte (Graphisme + 4 lignes de texte) est sélectionné, le nombre renvoyé concerne un caractère de la fenêtre de texte des 4 dernières lignes de l'écran.

Si $\text{\expra 2} + 16$ est compris entre 48 et 63, SCRN renvoie un nombre sans aucun rapport avec ce qu'il y a sur l'écran.

En mode texte (TEXT) SCRN retourne des nombres entre 0 et 15 dont la valeur est:

les 4 bits de poids fort (si \expra 2 est impair)

ou

les 4 bits de poids faible (si \expra 2 est pair) du caractère à la position ($\text{\expra 1} + 1$, INT (($\text{\expra 2} + 1$) (2))).

Donc l'expression $\text{CHR}\$ (\text{SCRN} (x-1, 2 * (y-1)) + 16 * \text{SCRN} (x-1, 2 * (y-1) + 1))$ renvoie le caractère sur l'écran à la position (x, y).

En mode graphique haute résolution, SCRN continue à fonctionner mais il renvoie les couleurs des rectangles du graphisme large résolution ou des caractères de la page texte. Le nombre renvoyé n'a aucun rapport avec ce que vous voyez sur l'écran haute résolution.

SCRN est interprété comme un mot réservé seulement si le premier caractère différent de l'espace suivant SCRN est une parenthèse ouvrante.

HGR imm et pg
HGR

● Lisez attentivement l'annexe A avant d'utiliser cette instruction.

Sans paramètres: Sélectionne le mode graphique haute résolution (360 sur 160 points) sur l'écran, laissant 4 lignes de texte en bas de l'écran. L'écran est nettoyé et la page 1 de la haute résolution s'affiche (de 8K à 16K).

HCOLOR n'est pas modifié par cette instruction.

La page texte (TEXT) n'est pas modifiée. L'utilisation de HGR ne réduit pas la fenêtre de texte, mais seules les 4 lignes du bas de l'écran sont visibles. Le curseur sera toujours dans la fenêtre de texte, mais il ne sera visible que s'il est déplacé sur une des 4 lignes du bas de l'écran.

Après l'exécution de HGR,

POKE -16302,0

ou bien

POKE 49239,0

convertissent l'écran en graphique haute résolution complet (360 sur 192 points) en supprimant les 4 lignes de texte.

Si vous refaites HGR après un des POKE ci-dessus, vous nettoyez l'écran et récupérez les 4 lignes de texte en bas de l'écran.



Si vous utilisez le mot réservé HGR comme premiers caractères d'un nom de variable, HGR risque de s'exécuter avant que le message:

?SYNTAX ERROR (erreur de syntaxe) s'affiche.

Par exemple, l'instruction:

```
HGRAS = 4
```

provoque une exécution non voulue de la commande HGR qui risque d'effacer votre programme.



Un programme très long s'étendant au-dessus de la case mémoire 8192 s'effacera partiellement si vous exécutez la commande HGR, ou risque d'afficher des points non désirés dans la page 1 de la haute résolution.

En particulier les chaînes de caractères sont stockées en haut de la mémoire, sur les configurations de 16K. Ces données risquent d'être placées en page 1 de la haute résolution. Fixez HIMEM: 8192 pour protéger votre programme et la page 1 de la haute résolution.

```
HGR2      imm et pg  
HGR2
```

Sans paramètres. Cette commande sélectionne la page 2 de la haute résolution sur l'écran (360 sur 192 points). Il n'y a pas 4 lignes de texte en bas de l'écran, tout est graphique. L'écran est nettoyé et la page 2 (de 16K à 24K) de la mémoire s'affiche. Cette page de mémoire (et évidemment la commande HGR2) n'est pas disponible sur une configuration de moins de 24K mémoire. Sur un système possédant cette configuration, utiliser HGR2 au lieu d'HGR permet de maximiser la place mémoire disponible au programme.

Sur les systèmes de 24K mémoire fixez HIMEM: 16384 pour protéger la page 2 de graphisme haute résolution et le programme (surtout les chaînes de caractères qui sont stockées en haut de la mémoire).



Si le mot réservé HGR2 est utilisé comme premiers caractères d'un nom de variable, la commande HGR2 sera exécuté avant que le message:

?SYNTAX ERROR (erreur de syntaxe) s'affiche sur l'écran.

Par exemple, en exécutant une instruction telle que:

```
140 IF X > 140 THEN HGR2BLA = 12
```

nettoie instantanément l'écran et une partie du programme risque de s'effacer.

La commande:

```
POKE -16301,0
```

fait apparaître 4 lignes de texte au bas de l'écran, cependant ces 4 lignes font partie de la page 2 de texte qui n'est pas facilement accessible à l'utilisateur.



Attention, sur un ITT 2020 de 32K sur lequel est branché un lecteur de disquette ITT, HGR2 détruit le programme de gestion de la disquette. Utilisez HGR.

HCOLOR imm et pg
 HCOLOR extra

Cette instruction définit la couleur du graphisme haute résolution, la couleur choisie est spécifiée par \extra\ qui doit être compris entre 0 et 7.

Voici la correspondance entre les couleurs et les codes:

0	1er noir	
1	vert (dépend du téléviseur)	violet
2	bleu (dépend du téléviseur)	vert
3	1er blanc	
4	2ème noir	
5	dépend du téléviseur	bleu
6	dépend du téléviseur	marron
7	2ème blanc	



Un point en haute résolution dessiné avec la couleur 3 (HCOLOR = 3) sera bleu si la coordonnée est pair en x, vert si la coordonnée (x, y) et (x + 1, y) sont dessinées. Cela est dû aux caractéristiques des TV du commerce.

HCOLOR n'est pas modifié par HGR, HGR2 ou RUN.

Avant qu'une instruction HCOLOR soit exécutée, la couleur des points dessinés est inconnue.

Utilisée en mode graphique large résolution, HCOLOR ne modifie rien à la couleur (COLOR).

H PLOT imm et pg
 H PLOT extra 1, extra 2
 H PLOT TO extra 3, extra 4
 H PLOT extra 1, extra 2 TO extra 5, extra 4

La première option dessine un point en haute résolution aux coordonnées $x = \text{\extra 1}$ et $y = \text{\extra 2}$.

La couleur du point est déterminée par la plus récente instruction HCOLOR exécutée.

La valeur d'HCOLOR est déterminée si elle n'a pas été préalablement spécifiée.

La deuxième option trace une ligne qui rejoint le DERNIER point dessiné sur l'écran au point de coordonnées (\extra 3 , \extra 4). La couleur de la ligne est celle du dernier point dessiné. Si aucun point n'a été préalablement dessiné, aucune ligne ne se trace.

Une troisième option trace une ligne joignant les points (\extra 1 , \extra 2) et (\extra 3 , \extra 4) dans la couleur dernièrement spécifiée par HCOLOR.

L'instruction:

H PLOT 0, 0 TO 357, 0 : H PLOT TO 357, 159 : H PLOT TO 0, 159 : H PLOT TO 0, 0

trace une bordure rectangulaire sur l'écran haute résolution.



H PLOT DOIT être précédé par HGR ou HGR2 pour éviter de perdre programme et variables.

\expra 1\ et \expra 3\ doivent être compris entre 0 et 357
 \expra 2\ et \expra 4\ doivent être compris entre 0 et 191

\expra 1\ et \expra 2\ peuvent être plus grands, égaux ou plus petits que \expra 3\ ou \expra 4\.

H PLOT en dehors des limites exigées provoque le message:

?ILLEGAL QUANTITY ERROR sur l'écran.

Si vous utilisez la haute résolution en mode mixte (graphisme + 4 lignes de texte), dessine des points d'ordonnées y entre 160 et 191 n'aura aucun effet sur l'écran.

PDL imm et pg
 PDL (expra)

Cette fonction retourne la valeur (entre 0 et 255) représentant la position du levier de commande spécifié par \expra\ ou \expra\ est compris entre 0 et 3.
 Un levier de commande est une résistance variant de 0 à 150K ohms.

Si deux leviers de commande sont lus consécutivement par l'instruction PDL, le nombre lu sur le second levier risque d'être affecté par le nombre lu sur le 1er levier de commande. Pour obtenir des lectures de meilleur précision, intercaler plusieurs lignes de programme entre les deux instructions PDL, ou bien intercalez une boucle d'attente comme:

```
FOR I = 1 TO 10 : NEXT I
```

Si \expra\ est négatif ou supérieur à 255, le message:

?ILLEGAL QUANTITY ERROR (quantité illégale, erreur) s'affiche sur l'écran.



Si \expra\ est compris entre 4 et 255, PDL renvoie un nombre imprévisible entre 0 et 255 qui risque de gêner la bonne exécution du programme.

Par exemple, si \expra\ est compris entre 204 et 269, l'instruction PDL est souvent accompagnée par un "clic" venant du haut-parleur lors de l'exécution.



Si \expra\ est entre 236 et 239, PDL (expra) risque d'effectuer un:

POKE -16540 + \expra\, 0

ce qui risque d'entraîner la sélection du mode graphique (GR) pour \expra\ = 236, la sélection du mode texte (TEXT) pour \expra\ = 237, etc.
 (Voir annexe J)

En plus du contrôle de 4 leviers de commande PDL le B.E.V.F. peut lire l'état de 3 boutons de jeu situés sur les leviers, en utilisant plusieurs commandes PEEK, pour en savoir plus, référez-vous à l'annexe J.

Notes



CHAPITRE 9

LES FIGURES HAUTE RESOLUTION

COMMENT CREER LA TABLE

DE CONSTRUCTION D'UNE FIGURE

Le B.E.V.F. possède 5 commandes spéciales qui vous permettent de manipuler sur l'écran haute résolution des figures que vous avez créées: DRAW, XDRAW, ROT, SCALE, SHLOAD.

Avant d'utiliser ces commandes, il faut construire une figure, comme si vous l'aviez dessinée sur papier, que l'on communique à la mémoire de l'ordinateur. La construction en mémoire de la figure se fait en codant en binaire chaque déplacement, chaque trait que vous avez effectué sur le papier pour dessiner la figure. Les informations, après avoir été traduites en binaire, sont stockées séquentiellement dans la mémoire de l'ITT 2020, et forment la table de construction des figures. Vous pouvez la conserver sur disquette ou cassette pour pouvoir ensuite la réutiliser.

On appelle "vecteur dessin" un déplacement unitaire d'un crayon (levé ou appuyé) sur une feuille de papier quadrillé. Chaque octet d'une table de construction est divisé en trois sections. Et chaque section représente un vecteur dessin, où est indiqué le sens de déplacement du crayon (haut, bas, droite, gauche) et spécifie si le crayon était appuyé sur le papier ou si le déplacement s'est fait crayon levé. Les instructions DRAW et XDRAW parcourent du premier octet au dernier octet, section par section, la table de construction pour reproduire la figure sur l'écran. Quand tous les bits d'un octet sont à zéro, la figure est considérée comme terminée par le B.E.V.F.

Voici comment se disposent les 3 sections A, B, C d'un octet dans une table de construction.

	octet		
Section	C	B	A
Numéro de bit	76	543	210
Désignation	SS	DSS	DSS

Chaque paire SS indique le Sens de déplacement et chaque bit D spécifie si le point doit être dessiné (crayon appuyé sur le papier) ou non (crayon levé) sur l'écran.

Si:

SS = 00	déplacement vers le haut
SS = 01	déplacement vers la droite
SS = 10	déplacement vers le bas
SS = 11	déplacement vers la gauche

Si:
 D = 0 ne pas dessiner
 D = 1 dessiner

Remarquez que la dernière section, C n'a pas de désignation D, pour cette section, D est toujours considéré comme nul. Donc la section C ne peut indiquer QUE des déplacements sans dessiner de points.

Chaque octet peut donc contenir 3 vecteurs dessin.
 Un dans la section A, un dans la section B et un troisième, de déplacement simple, dans la section C.

Les instructions DRAW et XDRAW parcourent les sections de droite à gauche (c'est-à-dire section A, puis B, puis C). Si une section d'un octet contient des zéros, alors cette section est ignorée par le B.E.V.F., c'est pourquoi la section C à 00 n'indique pas un déplacement vers le haut sans dessin car cette section étant à 00, sera ignorée. De même manière, si la section C est à 00 (ignorée) la section B ne peut être à 000 car sinon elle sera aussi ignorée. La section A à 000 terminera la table de construction sauf s'il y a au moins un bit à 1 dans les sections B ou C.

Supposons que sur l'écran vous vouliez dessiner cette figure:



Illustration 1

Dessinez-la d'abord sur une feuille de papier quadrillé. Puis en fixant le point de départ au centre, décrivez la figure sans lever la crayon du papier, vous obtiendrez ce dessin:

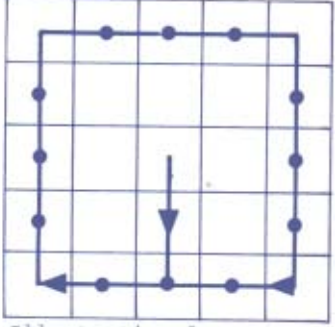


Illustration 2

N'oubliez pas que le crayon ne peut se déplacer obliquement mais doit forcément suivre l'une des quatre directions suivantes: haut, droite, bas, gauche.
 Maintenant redessinez la figure et introduisez les "vecteurs dessin", en différenciant bien les vecteurs qui indiquent: dessiner un point, puis se déplacer, et ceux qui indiquent: se déplacer sans dessiner de points.
 Vous obtiendrez l'illustration suivante, qui est en fait un remaniement de l'illustration 2.

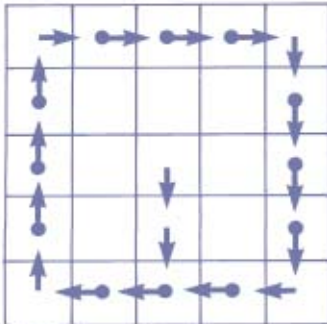


Illustration 3

Maintenant, "linéarisez" les vecteurs dessin pour les présenter sur une ligne, dans l'ordre où vous les avez dessinés. Vous obtenez:



Ensuite établissez la table de construction de l'illustration 5 en vous servant de l'illustration 4 qui vous rappelle la correspondance entre le vecteur dessin et sa valeur binaire dans les sections.

Vecteur	Code section A, B	Code section C
	$\emptyset\emptyset\emptyset$	X
	$\emptyset\emptyset 1$	$\emptyset 1$
	$\emptyset 1\emptyset$	$1\emptyset$
	$\emptyset 11$	11
	$1\emptyset\emptyset$	X
	$1\emptyset 1$	X
	$11\emptyset$	X
	111	X

Déplacement

Déplacement + dessin d'1 point

Illustration 4

X indique que le codage de ce vecteur est impossible en section C.

TABLE DE CONSTRUCTION

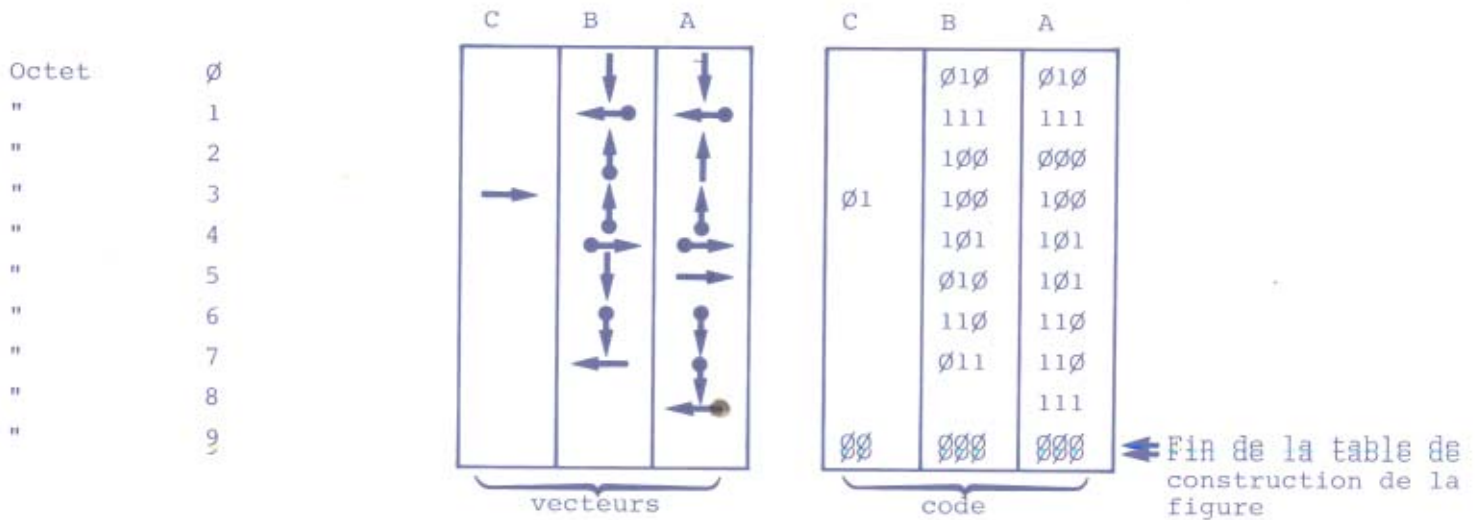


Illustration 5

Pour chaque vecteur sur la ligne, déterminez le code à l'aide de l'illustration 4 et placez ce code dans la première section LIBRE de la table.
 Si le code ne peut être placé dans la première section libre (par exemple: un vecteur dessin qui dessine un point suivi d'un déplacement ne peut être placé en section C), alors sautez cette section et placez-la dans la prochaine.
 Une fois que ce travail est terminé, vérifiez si tous les codages ont bien été faits.

Créez ensuite une autre table (illustration 6) qui est en fait une manière pratique de présenter la partie droite de l'illustration 5 pour la conversion en hexadécimal, l'hexadécimal étant le code d'entrée des données de la table de construction dans l'ordinateur.

		sections				
		C	B	A	recodés en hexadécimal	
Octet	∅	∅∅∅1		∅∅1∅	=	12
"	1	∅∅11		1111	=	3F
"	2	∅∅1∅		∅∅∅∅	=	2∅
"	3	∅11∅		∅1∅∅	=	64
"	4	∅∅1∅		11∅1	=	2D
"	5	∅∅∅1		∅1∅1	=	15
"	6	∅∅11		∅11∅	=	36
"	7	∅∅∅1		111∅	=	1E
"	8	∅∅∅∅		∅111	=	∅7
"	9	∅∅∅∅		∅∅∅∅	=	∅∅

Indique la fin de la table de construction

Illustration 6

Voici une table de conversion binaire/héxadécimal pour vous aider à établir des tables de construction:

Binaire	Héxadécimal
0000	= 0
0001	= 1
0010	= 2
0011	= 3
0100	= 4
0101	= 5
0110	= 6
0111	= 7
1000	= 8
1001	= 9
1010	= A
1011	= B
1100	= C
1101	= D
1110	= E
1111	= F

La table de valeurs en héxadécimal que vous avez finalement obtenue (illustration 6) est la représentation en mémoire de la figure dessinée. Mais il faut, avant d'introduire ces valeurs, donner des renseignements complémentaires au B.E.V.F. : le nombre de figures à dessiner et l'adresse relative de chacune des figures par rapport au premier octet de la table de construction. Cet ensemble s'appelle L'INDEX DE LA TABLE.

L'illustration 7 représente le cas général d'une table de construction complète des figures.

Pour notre exemple, l'index est simple, il n'y a qu'une figure dans la table.

L'adresse de début de la table de construction, que l'on appelle D doit contenir le nombre de figures en héxadécimal (entre 0 et FF c'est-à-dire entre 0 et 255 en décimal). Dans notre cas, ce nombre est 01. L'octet suivant (D+1) est inutilisé. Les octets suivants doivent, dans le cas général, contenir les adresses relatives aux débuts de chaque figure à dessiner. Mais dans notre cas, comme il n'y a qu'une figure, l'octet D+2 contiendra 04 et l'octet D+3 contiendra 00 (00 04 est en héxadécimal, l'adresse relative du début de la figure).

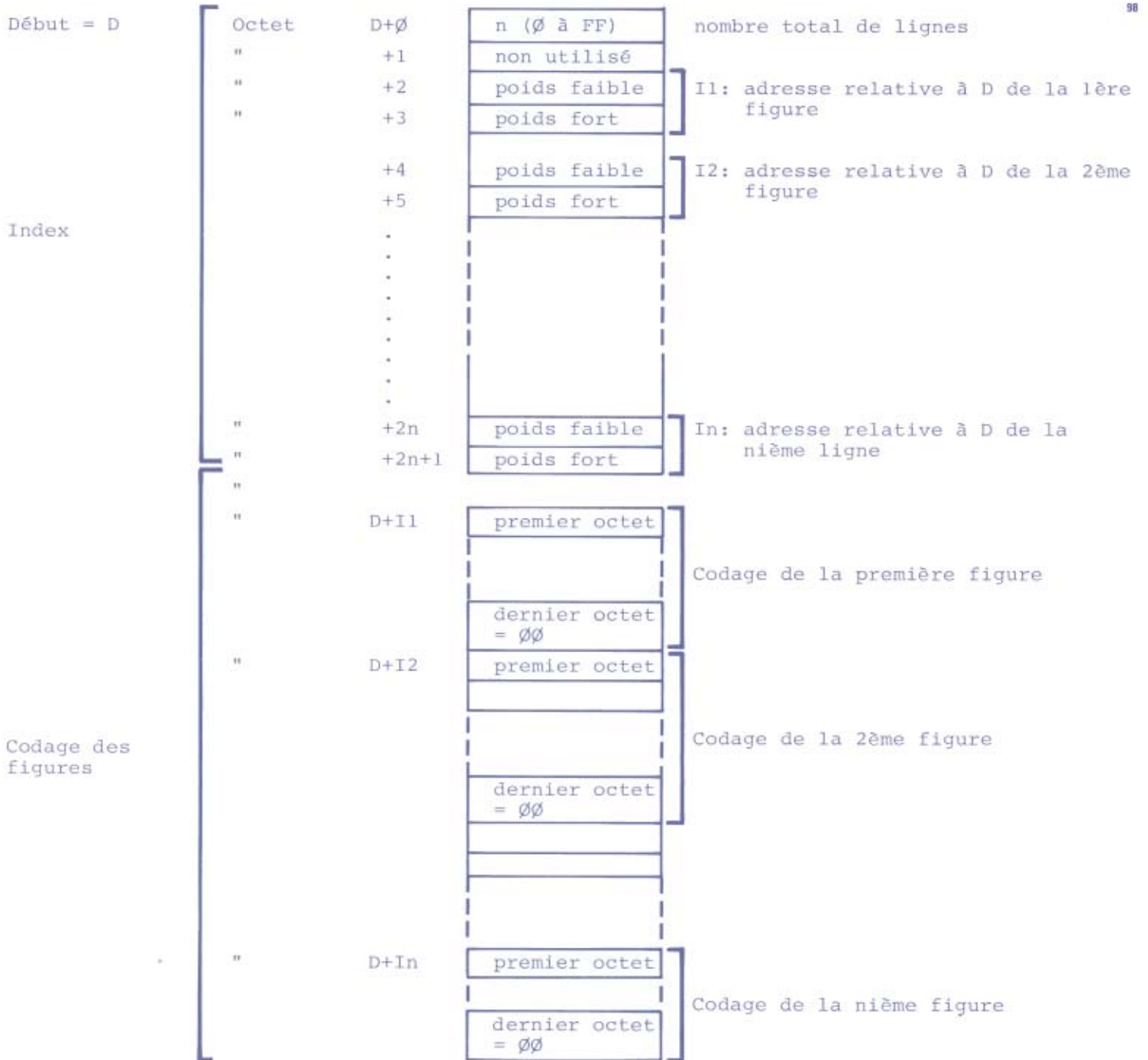


Illustration 7

L'illustration 8 représente la table de construction complète pour l'exemple de la figure traitée:

octet	Ø	Ø1	← nombre de figures
"	1	ØØ	} adresse relative au début du codage de la figure 1
"	2	Ø4	
"	3	ØØ	} codage de la figure 1
"	4	12	
"	5	3F	
"	6	2Ø	
"	7	64	
"	8	2D	
"	9	15	
"	A	36	
"	B	1F	} ← Fin de la figure
"	C	Ø7	
"	D	ØØ	

Illustration 8

Nous sommes maintenant prêts pour introduire la table de construction dans la mémoire de l'ITT 2Ø2Ø. En premier lieu, choisissons une adresse d'implantation de la table, pour cet exemple, nous choisirons 1DFC (Attention: l'adresse d'implantation doit être en-dessous de l'adresse maximum disponible pour la configuration, d'autre part faites attention de ne pas placer cette adresse dans les pages 1 et 2 de la haute résolution, car l'exécution de HGR et HGR2 effacerait la table).

Nous avons choisi 1DFC car elle est juste au-dessous de la page 1 de la haute résolution que l'on sélectionne par HGR.

Appuyez sur **RESET** pour rentrer en langage machine.

Pour stocker les données de la table à partir de l'adresse 1DFC, tapez sans oublier les espaces:

1DFC: Ø1 ØØ Ø4 ØØ 12 3F 2Ø 64 2D 15 36 1E Ø7 ØØ puis **RETURN**

Pour vérifier ce que vous avez tapé, faites:

1DFC **RETURN**

1DFC -Ø1

***RETURN**

ØØ Ø4 ØØ

***RETURN**

1E ØØ -12 3F 2Ø 64 2D 15 36 1E

***RETURN**

1E Ø8 -Ø7 ØØ XX XX XX XX XX XX

Les octets XX sont à ignorer.

Si, à la vérification votre table est incorrecte, réintroduisez-la.

Si elle est correcte, il ne reste plus qu'à communiquer au B.E.V.F. où se trouve l'adresse du début de la table de construction (cette opération se fait automatiquement quand vous utilisez l'instruction SHLOAD pour rappeler une table de construction conservée sur cassette).

Le B.E.V.F. considère l'adresse formée par le contenu des deux cases mémoire E8 et E9 comme l'adresse de début de la table de construction.

Dans notre exemple, l'adresse de début étant 1DFC tapez :

E8 : FC 1D **RETURN**

Pour protéger votre table d'une éventuelle destruction par un programme en B.E.V.F., fixez le HIMEM: (dont la valeur est fixée dans les cases mémoire 73 et 74 en hexadécimal) à l'adresse de début de la table:

73 : FC 1D

En utilisant l'instruction SHLOAD, cette opération est automatique.

Vous pouvez maintenant utiliser la figure pour la dessiner sur l'écran ou la conserver sur cassette.

CONSERVER LA TABLE DE CONSTRUCTION DES FIGURES

Pour conserver sur cassette une table de construction, vous avez besoin de connaître 3 renseignements:

- 1 - l'adresse de début de la table (1DFC dans notre exemple)
- 2 - l'adresse de fin de la table (1E00 dans notre exemple)
- 3 - la différence entre 2 et 1 (000D dans notre exemple)

La différence entre les adresses de fin et de début doit être stockée: poids faible dans la case mémoire 0 et poids fort dans la case mémoire 1.

Tapez pour notre exemple:

0 : 0D 00 **RETURN**

Vous pouvez maintenant conserver sur cassette la longueur de la table stockée aux adresses 0 et 1 et ensuite la table de construction elle-même stockée entre son adresse de début et son adresse de fin:

0. 1W 1DFC.1E09W

N'appuyez pas sur **RETURN** avant d'avoir fait tourner votre magnétophone en mode enregistrement. Quand le magnétophone enregistre, vous pouvez appuyer sur **RETURN**.

Pour rappeler la table de construction, cherchez l'enregistrement sur votre magnétophone et faites le tourner en mode lecture puis tapez au clavier:

SHLOAD **RETURN**

Vous entendrez un premier "bip" sonore indiquant que la longueur de la table a été lue par l'ordinateur, puis au second "bip" indiquant que la table de construction s'est chargée dans l'ordinateur.

UTILISER LA TABLE DE CONSTRUCTION DES FIGURES

Vous pouvez maintenant écrire des programmes en B.E.V.F. qui utiliseront les commandes DRAW, XDRAW, ROT et SCALE permettant d'exploiter les (la dans notre exemple) figures.

Voici un programme qui dessine la figure de notre exemple sur l'écran, la fait tourner de

16 degrés, la dessine de nouveau en augmentant la taille, la fait tourner de 16 degrés et ainsi de suite.

```
1Ø HGR
2Ø HCOLOR = 3
3Ø FOR R = 1 TO 5Ø
4Ø ROT = R
5Ø SCALE = R
6Ø DRAW 1 AT 139, 79
7Ø NEXT R
```

Pour ne voir qu'une seule figure, insérez la ligne:

```
65 END
```

Pour faire une pause et effacer chaque figure après qu'elle se soit dessinée, insérez ces lignes:

```
63 FOR I = Ø TO 1ØØØ : NEXT I
65 XDRAW 1 AT 139, 79
```

```
DRAW      imm et pg
DRAW      expra 1 AT expra 2, expra 3
DRAW      expra 1
```

Avec la première option DRAW dessine une figure en haute résolution, commençant au point de coordonnées $x = \backslash\text{expra } 2\backslash$ et $y = \backslash\text{expra } 3\backslash$.

La figure dessinée est la $\backslash\text{expra}\backslash$ ième figure de la table de construction des figures que l'on a, soit rappelée du magnétophone avec l'instruction SHLOAD, soit introduite soi-même au clavier en langage machine.

$\backslash\text{expra } 1\backslash$ doit être compris entre 0 et n ou n est le nombre total de figures que la table de construction comprend. (n compris entre 0 et 255)

$\backslash\text{expra } 2\backslash$ doit être compris entre 0 et 278.
 $\backslash\text{expra } 3\backslash$ entre 0 et 191.

Si un de ces trois intervalles de valeurs n'est pas respecté, le message:
?ILLEGAL QUANTITY ERROR (quantité illégale, erreur) s'affiche.

La couleur (HCOLOR) la rotation (ROT) et l'échelle (SCALE) doivent être spécifiées avant que DRAW soit exécuté.

La deuxième option a un effet similaire à la première mais le dessin de la figure commence à partir du dernier point dessiné par le dernier HPLOT, DRAW ou XDRAW exécuté.

Si un DRAW est exécuté alors qu'il n'y a pas de table de construction en mémoire, le système risque de boucler, ou bien des figures aléatoires risquent de se dessiner. Il est possible que votre programme s'efface. Pour récupérer le contrôle de l'ordinateur faites:

```
RESET      CTRL      RETURN
           C
```

```
XDRAW      imm et pg
XDRAW      expra 1 [AT expra 2, expra 3]
```

Commande similaire à DRAW excepté que la couleur de la figure est la couleur complémentaire de celle du dernier point dessiné.

Voici les paires de couleurs complémentaires:

```
blanc et noir
magenta et vert
```

XDRAW sert à facilement effacer les figures.

Si vous dessinez une figure (XDRAW) et que vous la redessinez ensuite (XDRAW) la figure disparaîtra sans effacer le fond de l'écran.



Voir la remarque de DRAW.

```
ROT        imm et pg
ROT        = expra
```

Fixe la rotation angulaire d'une figure qui sera dessinée par DRAW ou XDRAW. La rotation est fixée par \expra\, qui doit être entre 0 et 255.

ROT = 0 fixe l'orientation de la figure à celle que la table de construction prévoyait.

```
ROT = 16 fait tourner la figure de 90 degrés
ROT = 32 fait tourner la figure de 180 degrés.
```

Ainsi de suite...

La rotation est périodique, de période 64.

Pour SCALE = 1, 4 rotations différentes sont effectives 0, 16, 32, 48.

Pour SCALE = 2, 8 rotations différentes sont effectives et ainsi de suite...

Une rotation intermédiaire fera que la rotation effective sera celle de la valeur existante (pour l'échelle) en plus proche.

ROT est considéré comme un mot réservé seulement si le premier caractère différent d'un espace suivant ROT est le signe (=).

```
SCALE      imm et pg
SCALE      = expra
```

Fixe l'échelle de la figure qui sera dessinée par DRAW et XDRAW. Le facteur d'échelle SCALE varie entre 1 et 255.

1 est la taille de la figure telle qu'elle est conçue dans la table de construction.

255 est la taille où chaque vecteur dessin sera reproduit 255 fois plus grand qu'il n'avait été conçu.

REMARQUE: SCALE = 0 est la taille maximum.

SCALE est considéré comme un mot réservé seulement s'il est suivi par (=) comme

premier caractère différent de l'espace.

SHLOAD imm et pg
SHLOAD

Rappelle une table de construction des figures stockée sur cassette. La table est chargée dans l'ITT 2020 juste en-dessous de HIMEM: et HIMEM: est ensuite fixé juste en-dessous de la table pour la protéger. L'instruction SHLOAD communique automatiquement au B.E.V.F. l'adresse d'implantation de la table de construction.

Si une seconde table de construction est communiquée à l'ordinateur par l'intermédiaire des cassettes, remettez HIMEM: à sa valeur d'origine pour éviter de gaspiller de la place mémoire.

Les instructions pour conserver une table de construction des figures sur une cassette sont données en début de ce chapitre.

Sur une configuration de 16K, HGR nettoie les 8K de mémoire allant du 8ème au 16ème K de l'ordinateur.

Pour forcer SHLOAD à placer la table sous la page 1 de la haute résolution, fixez HIMEM: 8192 avant d'exécuter la commande SHLOAD.

Seul **RESET** peut interrompre SHLOAD.

Si un nom de variable commence par SHLOAD, la commande SHLOAD risque de s'exécuter avant que le message:

?SYNTAX ERROR (erreur de syntaxe) s'affiche.

L'instruction SHLOADER = 59

fait boucler le système, qui attend indéfiniment une table du magnétophone.

Faites **RESET** **CTRL** pour récupérer le contrôle de l'ordinateur.

C

Notes



CHAPITRE 10

QUELQUES FONCTIONS MATHÉMATIQUES

LES FONCTIONS DISPONIBLES AVEC LE B.E.V.F.

Toutes ces fonctions peuvent être utilisées dans un calcul arithmétique. Elles peuvent s'utiliser aussi bien en mode immédiat qu'en mode programme.

Voici une liste de fonctions mathématiques. Les autres fonctions ont été expliquées dans les autres chapitres.

SIN (expr)
donne le sinus d'\expr\ en radians.

COS (expr)
donne le cosinus d'\expr\ en radians.

TAN (expr)
donne la tangente d'\expr\ en radians.

ATN (expr)
donne l'arc-tangente en radians d'\expr\
L'angle donné est compris entre $-\pi/2$ et $+\pi/2$ radians.

INT (expr)
donne la partie entière d'\expr\.

RND (expr)
donne un nombre aléatoire plus grand ou égal à 0 mais plus petit que 1.
Si \expr\ est plus grand que zéro, RND (expr) donne à chaque fois un nombre aléatoire différent. Si \expr\ est inférieur à zéro, RND (expr) génère le même nombre aléatoire chaque fois qu'il est utilisé avec une \expr\, comme si le B.E.V.F. possédait un nombre aléatoire fixe en mémoire.
Si un nombre aléatoire est tiré avec \expr\ négatif, les nombres aléatoires tirés ensuite avec \expr\ positif suivront la même séquence chaque fois.

Une nouvelle séquence de nombres aléatoires est générée à chaque fois qu'un nombre aléatoire est tiré avec \expr\ négatif.

L'intérêt de tirer un nombre aléatoire avec un \expr\ négatif est de pouvoir initialiser une séquence répétable de nombres aléatoires. C'est particulièrement utile pour corriger les erreurs d'un programme utilisant RND.

Si \expr\ vaut 0, RND (expr) redonne le dernier nombre aléatoire tiré (CLEAR et NEW ne l'affectent pas). C'est parfois utile pour éviter d'avoir à assigner un nombre aléatoire dans une variable pour le conserver.

SGN (expr)
donne -1 si \expr\ < 0, donne 0 si \expr\ = 0 et donne +1 si \expr\ > 0.

ABS (expr)
donne la valeur absolue d'\expr\, c'est-à-dire \expr\. Si \expr\ >= 0 ou -\expr\ si \expr\ < 0.

SQR (expr)
donne la racine carrée d'\expr\. L'exécution est plus rapide que de faire $\wedge .5$

EXP (expr)
donne l'exponentielle (en base e) de \expr\.

LOG (expr)
donne le logarithme népérien de \expr\.

Notes

ANNEXE A :

1ÈRE PARTIE

MISE EN ROUTE DU B.E.V.F.

Le B.E.V.F. existe en deux versions.

La première sur carte ROM que l'on connecte sur l'ITT 2020. En actionnant un interrupteur et en tapant sur deux touches du clavier, l'ITT 2020 peut travailler en B.E.V.F. Il y a une économie de mémoire (le B.E.V.F. occupe 10K) et de temps (le chargement est immédiat) par rapport à la deuxième version qui est une version cassette que l'on doit stocker dans la mémoire RAM de l'ITT 2020 à partir du magnétophone.

Si vous utilisez la version cassette du B.E.V.F., voyez la deuxième partie de cette annexe pour des précautions spéciales et les différences entre les deux B.E.V.F.

IMPORTANT: le caractère de reconnaissance est un caractère propre au langage utilisé. Il permet en un coup d'oeil de savoir dans quel langage on travaille.

Jusqu'à présent vous connaissiez les caractères de reconnaissance:

* pour le langage machine

> pour le BASIC entier

en voici un troisième:

] pour le B.E.V.F.

Le simple fait de regarder le caractère de reconnaissance vous renseigne sur la nature du langage.

DIFFÉRENCES ENTRE LA CARTE " ROM " B.E.V.F. ET LA VERSION SUR CASSETTE

Installation:

la carte ROM du B.E.V.F. se connecte simplement à l'intérieur de l'ITT 2020. Vous devez cependant faire attention et suivre scrupuleusement les instructions ci-dessous:

- 1 - Déconnecter l'ITT 2020, très important pour éviter d'endommager l'appareil.
- 2 - Enlever le couvercle de l'ITT 2020 en tirant vers le haut la partie arrière du couvercle tout en tenant le boîtier de l'ordinateur.
Une fois que les deux clips sont dégagés, faites glisser le couvercle vers l'arrière jusqu'à ce que l'ordinateur soit dégagé.
- 3 - Il y a à l'intérieur de l'ITT 2020, à l'arrière de la carte des circuits, une rangée de 8 connecteurs d'entrée/sortie que l'on appelle ports E/S.
Le plus à gauche (en regardant l'ordinateur face au clavier) est le port d'E/S n° 0
Le plus à droite est le port d'E/S n° 7.
Tenir la carte B.E.V.F. de telle sorte que l'interrupteur soit tourné vers l'arrière de l'ordinateur. Insérez les contacts de la carte du B.E.V.F. dans le port E/S n° 0 (le plus à gauche).

Les contacts doivent être introduits dans le port E/S n° 0 avec une certaine force. La carte B.E.V.F. doit être placée dans le port E/S n° 0.

- 4 - Si la carte est bien placée, l'interrupteur de la carte B.E.V.F. devra être inséré dans une ouverture du boîtier qui permet de le manipuler par l'extérieur, couvercle posé.
- 5 - Remplacez le couvercle de l'ordinateur en le glissant par l'arrière et ensuite en pressant sur les deux coins arrière du couvercle de manière à engager les clips.
- 6 - Mettre sous tension l'ITT 2020.

Avec l'interrupteur de la carte B.E.V.F. en bas, l'ordinateur permet de travailler en BASIC Entier quand vous tapez **CTRL** (Tenir enfoncée la touche **CTRL** pendant que vous

appuyez sur **B**). Vous verrez le caractère de reconnaissance > qui indique que vous êtes en BASIC Entier.

Avec l'interrupteur en position haute, l'ordinateur permet le travail en B.E.V.F., pour cela tapez **RESET CTRL**. Le caractère de reconnaissance] indique que vous êtes en B.E.V.F.

Quand vous utilisez le DISQUE ITT, l'ordinateur choisira lui-même le B.E.V.F. ou le BASIC Entier, selon ses besoins.

Pas besoin de manipuler l'interrupteur.

Vous pouvez aussi sélectionner le B.E.V.F. sans toucher à l'interrupteur, faites:

RESET C080 **RETURN**
CTRL **RETURN**
B

Ou bien passez en BASIC Entier en faisant:

RESET C081 **RETURN**
CTRL **RETURN**
B

IMPORTANT: Si vous tapez accidentellement ou involontairement **RESET**, alors vous êtes en langage machine. Si vous voulez récupérer votre programme intact en B.E.V.F. tapez:

CTRL **RETURN**
C

2ÈME PARTIE : LE B.E.V.F. EN CASSETTE

B.E.V.F. est fourni avec l'ITT 2020 sur une cassette de magnétophone. Le B.E.V.F. utilisant 10K de mémoire RAM, il faut posséder un ITT 2020 dont la configuration mémoire est d'au moins 16K pour pouvoir utiliser ce langage.

POUR CHARGER LE B.E.V.F. EN MEMOIRE

Suivez les instructions ci-dessous, en connectant votre magnétophone à l'ordinateur.

- 1 - Pour commencer, tapez **RESET** **CTRL** pour sélectionner le BASIC Entier. Si cette procédure ne vous est pas familière, référez-vous au manuel du BASIC Entier. Vous saurez si vous êtes en BASIC Entier en voyant sur l'écran le caractère de reconnaissance > suivi du curseur clignotant.
- 2 - Placez la cassette du B.E.V.F. dans le magnétophone en utilisant la touche de recul rapide du magnétophone, mettez la bande à son début.
- 3 - Tapez LOAD.
- 4 - Faites tourner le magnétophone en mode lecture.
- 5 - Appuyez sur la touche **RETURN** de l'ordinateur. Le curseur disparaîtra et au bout de 15 à 20 secondes un "bip" sonore se fera entendre, c'est le signal de début de programme pour l'ordinateur.
Après une minute à 1 minute et demie d'attente, un second "bip" sonnera à nouveau et le curseur ainsi que le signe de reconnaissance du BASIC Entier réapparaîtront.
- 6 - Arrêtez le magnétophone, le B.E.V.F. est maintenant en mémoire de l'ITT 2020.
- 7 - Tapez RUN puis appuyez sur la touche **RETURN**. Sur l'écran apparaîtra alors une notice et le caractère de reconnaissance du B.E.V.F.:].

ATTENTION: Si accidentellement ou involontairement vous tapez la touche **RESET**, vous passez en langage machine. Pour récupérer le B.E.V.F. et le programme que vous utilisez, tapez:

0G **RETURN**

Si cela ne marche pas, vous avez à recharger le B.E.V.F. dans la mémoire de l'ITT 2020 pour pouvoir réutiliser le langage.

Dans ce manuel, **RESET** veut dire: enfoncer la touche marquée RESET, **RETURN** veut dire enfoncer la touche marquée **RETURN** et **CTRL** veut dire enfoncer et tenir la touche **CTRL** pendant que la touche **B** est appuyée.

La version du B.E.V.F. sur cassette ne fonctionne pas exactement comme celle sur carte ROM.

Il y a très peu de différences mais les instructions et commandes décrites dans ce manuel s'adressent aux utilisateurs de la carte ROM B.E.V.F.

Voici des explications sur ces différences.

La version cassette du B.E.V.F. utilisant 10K de mémoire utilisateur, cette version peut être utilisée sur des ordinateurs de moins de 16K.

Si la version cassette est chargée en mémoire, la première case mémoire disponible pour l'utilisateur est la case 12300. Le B.E.V.F. sur carte ROM n'étant pas résident en mémoire RAM, toute la mémoire est alors disponible.



HGR n'est pas utilisable sur la version cassette du B.E.V.F. L'exécution de HGR provoque le "nettoyage" de la page 1 de la haute résolution. C'est-à-dire de la zone mémoire allant de 8K à 16K. Comme la version cassette du B.E.V.F.

utilise une partie de cette zone, l'exécution de HGR détruirait une partie du B.E.V.F., effacerait votre programme et vous seriez obligé de recharger le B.E.V.F. au magnétophone.

Pour faire de la haute résolution et si votre ITT 2020 a une configuration d'au moins 24K utilisez la commande HGR2 qui vous sélectionne la page 2 du graphisme couleur haute résolution.

La commande POKE -16301,0 convertit l'écran haute résolution en mode mixte haute résolution (graphisme + 4 lignes de texte en bas de l'écran).

Cependant les 4 lignes de texte sont prises sur la page 2 de la mémoire texte. Sur la version cassette du B.E.V.F., la page 2 du texte est occupée par le B.E.V.F. lui-même.

Donc, avec une version cassette, le mode mixte graphique haute résolution + texte n'est pas utilisable.

En BASIC Entier et sur la version carte ROM du B.E.V.F. vous pouvez récupérer le programme après un **RESET** accidentel ou volontaire, en tapant:

CTRL
C **RETURN**

N'utilisez pas **CTRL** **RETURN** avec la version cassette du B.E.V.F., car vous perdrez

C

vos programmes et le B.E.V.F.

Pour récupérer votre programme avec la version cassette, faites:

ØG **RETURN**

EN RESUME: partout dans ce manuel où l'on indique **CTRL** **RETURN**, l'utilisateur de la version sur cassette du B.E.V.F. devra faire ØG RETURN à la place.

Quand on indique dans ce manuel:

CTRL **RETURN**
B

si vous exécutez cette commande avec la version cassette vous perdrez le B.E.V.F. et le programme en mémoire.

Avec la version cassette du B.E.V.F. faites CALL 11246 (au lieu de CALL 62450) pour nettoyer l'écran haute résolution HGR2.

Faites CALL 11250 (au lieu de CALL 62454) pour "peindre" l'écran haute résolution HGR2 de la couleur (HCOLOR) du dernier point dessiné (HPLOT).

Si vous exécutez ces commandes avant d'avoir exécuté HGR2 au moins une fois, ces appels de sous-programmes (CALL) risquent de détruire le B.E.V.F. de la mémoire.

ANNEXE B :

L'ÉDITION DES PROGRAMMES

Tout le monde fait des fautes... et surtout quand on écrit un programme pour ordinateur. Pour faciliter la correction de ces "inattentions" ITT a incorporé de nombreuses facilités d'édition au B.E.V.F.

Pour utiliser ces facilités, il faut que vous soyez vous-même familiarisé avec les fonctions de certaines touches du clavier de l'ITT 2020.

Il y a la touche **ESC**, la touche répétition **REPT**, les touches "flèche à droite" et "flèche à gauche".

ESC

C'est la touche la plus à gauche de la deuxième rangée en partant du haut du clavier. On l'utilise toujours avec une autre touche (telle que **A**, **B**, **C** ou **D**) et de la manière suivante: l'alternance; appuyez sur la touche **ESC**, la relâcher, appuyer sur la touche **A** (par exemple) et la relâcher.

Appuyer sur la touche **ESC** puis sur la touche **A** s'écrit:

ESC
A

Il y a quatre fonctions **ESC** pour l'édition:

ESC
A

 déplace le curseur vers la droite

ESC
B

 déplace le curseur vers la gauche

ESC
C

 déplace le curseur vers la bas

ESC
D

 déplace le curseur vers le haut

L'utilisation de la touche **ESC** et de la touche désirée permet de déplacer le curseur sur l'écran sans modifier le contenu de l'écran ni la mémoire de l'ordinateur.

"FLECHE A DROITE"

La flèche à droite déplace le curseur sur la droite. C'est une touche très utile, car non seulement elle déplace le curseur, mais en plus elle recopie dans la mémoire de l'ordinateur TOUS les caractères traversés par le curseur lors de son utilisation. Comme si vous aviez tapé les caractères du clavier. L'écran n'est pas modifié par la flèche à droite.

"FLECHE A GAUCHE"

La flèche à gauche déplace le curseur vers la gauche. A chaque déplacement, le caractère traversé est EFFACE de la ligne de programme que vous éditez. L'écran n'est pas modifié par l'utilisation de la flèche à gauche.

La flèche à gauche ne peut placer le curseur sur la première colonne de l'écran. Utilisez

ESC
B

 pour cela.

REPT

Utilisée avec une autre touche du clavier,

REPT

 permet la répétition du caractère frappé, aussi longtemps que les deux touches restent enfoncées. Utilisée en conjonction avec la "flèche à droite", permet de faciliter le travail de l'édition.

Vous savez maintenant comment utiliser les fonctions d'édition, voyons maintenant quelques exemples d'utilisation:

EXEMPLE 1 : MODIFIER DES CARACTÈRES DANS UNE LIGNE DE PROGRAMME

Supposons que vous avez introduit un programme et quand vous l'exécutez l'ordinateur affiche:

```
?SYNTAX ERROR IN XXXX (erreur de syntaxe en ligne XXXX) et l'ordinateur s'arrête en affichant ] et le curseur sur l'écran.
```

Voici la méthode pour remédier à ce genre de problèmes.

Tapez le programme suivant et exécutez le (RUN).

Notez bien que le "PRINT" et "PREGRAMME" sont des erreurs volontaires!

```
]10 PRINT "C'EST UN PREGRAMME"
```

```
]20 GOTO 10
```

```
]RUN
```

```
]?SYNTAX ERROR IN 10
```

```
]■
```

Tapez LIST puis

RETURN

```
]LIST
```

```
10 PRINT "C'EST UN PREGRAMME"
```

```
20 GOTO 10
```

```
]■
```

La ligne 10 doit être modifiée. Pour placer le curseur en début de ligne 10 il faut taper trois fois

ESC
D

 et une fois

ESC
B

.

Il est impératif d'utiliser

ESC
B

 pour placer le curseur sur le premier chiffre du numéro de ligne à modifier.

L'écran de votre TV ressemblera à ceci:

```
]LIST
```

```
10 PRINT "C'EST UN PREGRAMME"
```

```
20 GOTO 10
```

```
]■
```

Tapez maintenant 6 fois sur la "flèche à droite" pour amener le curseur sur la lettre M du mot PRINT.

N'oubliez pas que les caractères traversés par le curseur ont été recopiés dans la mémoire de l'ordinateur comme si vous les aviez tapés au clavier. L'écran ressemble maintenant

à ceci:

```
]LIST
10 PRINT "C'EST UN PREGRAMME"
20 GOTO 10
]
```

Tapez maintenant N pour changer la lettre M et pour avoir la syntaxe correcte PRINT, puis copiez les caractères à l'aide de la "flèche à droite" et de **REPT** jusqu'à la lettre E de PREGRAMME.

Maintenant votre écran ressemble à ceci:

```
LIST
10 PRINT "C'EST UN PRGRAMME"
20 GOTO 10
]
```

Si vous avez été trop loin avec la "flèche à droite" utilisez la "flèche à gauche" pour revenir sur le E.

Tapez maintenant la lettre O pour changer PREGRAMME en PROGRAMME puis copiez à l'aide de la "flèche à droite" et **REPT** le reste de la ligne 10. Finalement stockez la nouvelle ligne en mémoire à l'aide de la touche **RETURN**.

Tapez LIST pour voir le programme corrigé:

```
]LIST
10 PRINT "C'EST UN PROGRAMME"
20 GOTO 10
]
```

Exécutez (RUN) maintenant le programme (en utilisant **CTRL** pour l'arrêter).

```

      RUN
C'EST UN PROGRAMME
"  "  "
.
.
.
.
"  "  "
C'EST UN PROGRAMME
BREAK IN 10
]
```

exemple 2 : INSÉRER DANS UNE LIGNE DE PROGRAMME

Si dans l'exemple précédent vous voulez insérer une instruction TAB (10) après le PRINT en ligne 10, voici comment faire:

LISTer la ligne devant être modifiée

```
]LIST 10
10 PRINT "C'EST UN PROGRAMME"
]
```

A l'aide de

ESC
D

 et

ESC
B

 placez le curseur sur le premier chiffre du numéro de la ligne.

Puis recopiez l'instruction (avec la flèche à droite et

REPT

) jusqu'au premier guillemet. (Souvenez-vous qu'un caractère n'est copié en mémoire que si le curseur a traversé ce caractère et qu'il est placé sur le caractère immédiatement à droite).
Votre écran devrait afficher:

```
]LIST 10
10 PRINT " C'EST UN PROGRAMME"
]
```

Maintenant tapez un autre

ESC
D

 pour placer le curseur sur une ligne vide juste au-dessus de la ligne 10. Voici à quoi ressemble l'écran:

```
LIST 10
10 PRINT "C'EST UN PROGRAMME"
]
```

Tapez les caractères à insérer (TAB(10); dans ce cas). L'écran affiche maintenant:

```
]LIST 10
      TAB (10);
10 PRINT "C'EST UN PROGRAMME"
]
```

Tapez ensuite

ESC
C

 une fois pour que le curseur soit positionné comme ceci:

```
]LIST 10
      TAB (10);
10 PRINT "C'EST U"
]
```

Reculer maintenant avec

ESC
B

 jusqu'au premier guillemet (n'utilisez pas la flèche à gauche qui effacerait les caractères que vous venez de taper).
Voici ce que vous devez voir sur votre TV:

```
]LIST 10
      TAB (10);
]PRINT " C'EST UN PROGRAMME"
]
```

Enfin copiez le reste de la ligne avec la flèche à droite et la touche

REPT

.
L'écran se présentera ainsi:

```
]LIST 10
      TAB (10);
]PRINT "C'EST UN PROGRAMME"
]
```

Tapez

RETURN

 puis LIST pour obtenir:

```
LIST
10 PRINT TAB (10); "C'EST UN PROGRAMME"
20 GOTO 10
] =
```

Si vous recopiez une ligne formatée par un LIST, vous risquez d'introduire des espaces supplémentaires. Par exemple, si vous recopiez la ligne 10 ci-dessus, vous allez introduire des espaces non désirés entre O et G. Pour éviter cela, utilisez

ESC
A

.

ESC
A

 déplace

le curseur vers la droite SANS recopier les caractères. C'est surtout utile avec les instructions PRINT, REM et INPUT, car le B.E.V.F. n'ignore pas les espaces supplémentaires dans les instructions.

N'oubliez pas qu'avec les touches

ESC

, vous pouvez copier et éditer du texte affiché n'importe où sur l'écran.

Notes

MESSAGES D'ERREURS

Si une erreur se produit, le B.E.V.F. rend le contrôle de l'ordinateur à l'utilisateur, c'est-à-dire que l'exécution d'un programme s'interrompt et le caractère de reconnaissance] s'affiche, suivi du curseur. Les valeurs des variables et le programme ne sont pas modifiés, mais le programme ne peut être CONTInué et tous les compteurs de FOR...NEXT et GOSUB sont remis à 0.

Pour éviter cet arrêt de programme, utilisez l'instruction ONERR GOTO avec un programme de manipulation des erreurs.

Lorsqu'une erreur se produit en mode immédiat il n'y a pas d'indication de numéro de ligne.

Voici les formats des messages d'erreurs:

mode immédiat ?XX ERROR
mode programme ?XX ERROR IN YY

où XX est le type d'erreur et YY est le numéro de ligne où s'est produite l'erreur. Les erreurs en mode programme sont détectées au moment de l'exécution de l'instruction qui donne l'erreur.

Voici la liste des erreurs possibles et leur signification:

CAN'T CONTINUE (impossible de continuer)

Vous avez essayé de CONTInuer: l'exécution d'un programme inexistant en mémoire, ou bien le programme s'était arrêté sur une erreur, ou bien vous avez intercalé ou détruit une ligne.

DIVISION BY ZERO (division par zéro)

Division par zéro dans un calcul.

ILLEGAL DIRECT (mode immédiat illégal)

Vous ne pouvez pas utiliser les instructions INPUT, DEF FN, DATA, GET en mode immédiat.

ILLEGAL QUANTITY (quantité illégale)

Une opération arithmétique ou une chaîne de caractères a généré une quantité illégale; cela peut être dû à:

A - un index de tableau négatif ($A(-1) = X$)

B - utilisation d'une quantité négative ou nulle pour la fonction LOG ou négative pour la fonction SQR

C - $A \wedge B$ avec A négatif et B non entier

D - utilisation de MID\$, LEFT\$, RIGHT\$, WAIT, PEEK, POKE, TAB, SPC, ON...GOTO, ou toute instruction graphique avec de mauvais arguments.

NEXT WITHOUT FOR (NEXT sans FOR)

La variable de l'instruction NEXT ne correspond pas à la variable de l'instruction FOR, ou bien NEXT ne correspond pas au FOR en exécution.

OUT OF DATA (plus de données)

Une instruction READ a été exécutée mais toutes les données DATA ont déjà été lues. Le programme essaye de lire des données supplémentaires qui n'existent pas.

OUT OF MEMORY (plus de mémoire)

Tout ce qui suit peut être la cause de l'erreur:

- programme trop long
- des boucles FOR imbriquées sur plus de 10 niveaux
- plus de 24 niveaux de sous-programmes (GOSUB) imbriqués
- des expressions trop complexes
- plus de 36 niveaux de parenthèses imbriquées
- une diminution de LOMEM: (plus bas que sa dernière valeur)
- une modification de LOMEM: (trop haute)
- une modification de HIMEM: (trop basse)

FORMULA TOO COMPLEX (formule trop complexe)

Plus de deux instructions de type IF "XX" THEN sont exécutées.

OVER FLOW (dépassement de capacité)

Le résultat d'un calcul dépasse la capacité de traitement des nombre en B.E.V.F.

REDIM'D ARRAY (tableau redimensionné)

Un tableau était déjà dimensionné, lorsqu'une nouvelle instruction de dimension concernant le même tableau s'exécute. Cette erreur arrive souvent si un tableau est déjà utilisé sans avoir préalablement été dimensionné. Par exemple, si l'ordinateur exécute $A(4) = 3$ puis plus tard l'instruction $DIM A(100)$ s'exécute. Le message d'erreur peut être utilisé si vous désirez savoir à quelle ligne un tableau a été dimensionné: insérez une ligne au début du programme qui dimensionne ce même tableau. Quand vous exécuterez le programme, l'erreur se produira à la ligne où le tableau était initialement dimensionné.

RETURN WITHOUT GOSUB (RETURN sans GOSUB)

Une instruction RETURN est rencontrée alors qu'il n'y avait pas de GOSUB correspondant.

STRING TOO LONG (chaîne trop longue)

Lors d'une création ou d'une concaténation de chaîne de caractères, la longueur maximale de 255 caractères n'a pas été respectée.

BAD SUBSCRIPT (erreur d'index)

Un index de tableau ne correspondant pas à la dimension déclarée a été utilisé. Cette erreur peut se produire si un tableau s'utilise avec un nombre de dimensions différent de celui déclaré.

Par exemple: $A(1, 1, 1) = 2$ alors que vous avez fait $DIM A(2, 2)$.

SYNTAX (syntaxe)

S'il manque des parenthèses dans une expression, si un caractère illégal apparaît dans une ligne, si la ponctuation est incorrecte, etc.

TYPE MISMATCH (erreur de type)

La partie gauche d'une assignation est une expression arithmétique alors que la partie droite est une chaîne ou vice-versa. Ou une fonction travaillant avec un argument alphanumérique est utilisé avec une expression arithmétique, ou inversement.

UNDEF'D STATEMENT (instruction non définie)

Le numéro de ligne suivant un GOTO, GOSUB ou THEN n'existe pas dans le programme.

UNDEF'D FUNCTION (fonction non définie)

L'appel d'une fonction non définie par l'utilisateur provoque cette erreur.

Notes

ANNEXE D :

COMMENT GAGNER DE LA PLACE MÉMOIRE

Voici des "trucs" pour gagner de la place mémoire sur des programmes. Les paragraphes 1 et 2 ne sont à considérer que si vous êtes vraiment limité par la place mémoire.

Les programmeurs professionnels conservent en général deux versions de leurs programmes: une fortement commentée par les instructions REM et l'autre contractée pour utiliser le minimum de mémoire.

- 1 - Utilisez les séparateurs (:) pour écrire plusieurs instructions sur une même ligne; en effet la déclaration d'une ligne occupe 5 octets: 2 pour le numéro de la ligne (que le numéro soit 1 ou 65529) et 3 pour la gestion de la ligne par le B.E.V.F. En écrivant le maximum d'instructions sur une même ligne, vous récupérez la place qui aurait été nécessaire à la déclaration de chaque ligne. Une ligne peut contenir jusqu'à 239 caractères.

REMARQUE: combiner plusieurs instructions sur la même ligne complique les éventuelles éditions et la lecture du programme, pour les autres, et même pour vous, si vous replongez dans le programme après un certain temps.

- 2 - Supprimez toutes les indications REM. Chaque REMarque utilise au moins 1 octet, plus 1 octet pour chaque caractère de la remarque.

Par exemple: l'instruction
130 REM CECI EST UN COMMENTAIRE. occupe 31 octets en mémoire.

REMARQUE: comme pour les instructions multiples sur une même ligne, le fait de supprimer les REM complique fortement la lisibilité du programme.

- 3 - Utilisez autant que possible des tableaux de nombres entiers plutôt que des tableaux de nombres réels. (Voir plus loin dans cette annexe: "PLACE MEMOIRE UTILISEE")
- 4 - Utilisez des variables plutôt que des constantes.
En effet, si dans un programme vous utilisez dix fois la constante 3.14159, vous aurez intérêt à insérer la ligne suivante:
10 PI = 3.14159
et remplacer dans tout le programme 3.14159 par PI à chaque fois que vous rencontrerez cette constante. Vous gagnerez 40 octets et le programme tournera plus vite.
- 5 - Un programme n'a pas besoin d'être terminé par un END, donc une instruction END peut être supprimée en fin de programme.
- 6 - Réutilisez les mêmes variables dans un programme. Si par exemple, la variable est une variable utilisée localement dans un programme vous pouvez la réutiliser dans une autre partie du programme pour une fonction totalement différente.
Ou si votre programme pose plusieurs fois des questions dont les réponses sont du type OUI, du type NON vous pouvez réutiliser à chaque fois la même variable alphanumérique pour stocker la réponse de l'utilisateur.
- 7 - Essayez d'utiliser un maximum de sous-programmes (GOSUB)

- 8 - Utilisez les éléments d'index \emptyset des tableaux par exemple A (\emptyset) B (1, \emptyset)
- 9 - Quand A\$ = "CHAT" est changé en A\$ = "CHIEN" l'ancienne valeur CHAT de A\$ n'est pas effacée de la mémoire. En utilisant périodiquement une instruction comme X = FRE (\emptyset) le B.E.V.F. fait un "nettoyage maison" de sa mémoire et détruit les anciennes valeurs des variables alphanumériques.

PLACE MÉMOIRE UTILISÉE

Les variables simples: réelles, entières ou alphanumériques telles que V, V% ou V\$ utilisent 7 octets en mémoire.

Une variable réelle utilise 2 octets pour le nom de la variable et 5 octets pour la valeur. (4 pour la mantisse et 1 pour l'exposant)

Une variable entière utilise 2 octets pour le nom de variable, 2 octets pour la valeur et les 3 octets restants sont à \emptyset .

Une variable alphanumérique utilise 2 octets pour le nom de variable, 1 octet pour la longueur de sa chaîne, 2 octets pour un pointeur d'adresse de la chaîne et les 2 octets restants sont à \emptyset .

Un tableau réel utilise un minimum de 12 octets: 2 pour le nom du tableau, 2 pour la taille du tableau, 1 pour le nombre de dimensions, 2 pour le nombre d'éléments de chaque dimension et 5 pour chaque élément.

Un tableau entier n'utilise que 2 octets pour chaque élément.

Un tableau de chaîne de caractères utilise 3 octets pour chaque élément: 1 pour la longueur de l'élément et 2 pour le pointeur d'adresse.

Les variables de chaînes ou de tableaux de chaînes utilisent 1 octet pour chaque caractère de chaîne. Les chaînes sont stockées à partir des HIMEM: dans l'ordre où elles apparaissent dans le programme. Quand une nouvelle fonction est définie par une instruction DEF, 6 octets sont occupés par le pointeur de la fonction.

Tous les mots réservés tels que FOR, SIN, utilisent 1 octet en mémoire. Tous les autres caractères utilisent 1 octet dans le stockage du programme.

À l'exécution d'un programme, l'espace mémoire s'alloue dynamiquement sur les piles comme suit:

- 1 - Chaque boucle FOR...NEXT en exécution occupe 16 octets.
- 2 - Chaque GOSUB en exécution occupe 6 octets
- 3 - Chaque parenthèse dans une expression en cours de calcul utilise 4 octets et le résultat partiel utilise 12 octets.



Notes

ANNEXE E :

POUR ACCÉLÉRER LA VITESSE D'EXÉCUTION DE VOS PROGRAMMES

Voici quelques "tuyaux" pour améliorer le temps d'exécution de vos programmes. Quelques unes de ces astuces sont similaires à celles employées pour réduire la place mémoire occupée par le programme.

Cela veut dire qu'en général si vous réduisez l'encombrement d'un programme, vous en accélérez l'exécution.

- 1 - Utiliser des variables au lieu de constantes est une astuce qui permet parfois de MULTIPLIER PAR 10 la vitesse d'exécution.
Il faut beaucoup plus de temps à l'interpréteur de l'ITT pour convertir une constante dans sa représentation interne réelle que pour aller chercher la valeur à l'intérieur d'une variable ou d'un tableau. L'effet est d'autant plus remarquable si la constante était utilisée dans des boucles.
- 2 - Les variables sont stockées dans la table des variables selon leur ordre d'apparition dans le programme.
Cela veut dire qu'à l'exécution d'un programme:

$$\emptyset A = \emptyset : B = A : C = A$$
 placera dans la table des variables A en premier, B en second et C en troisième. Si plus tard, pendant l'exécution, le programme a besoin de la variable A, il n'aura à faire qu'une recherche dans la table des variables. S'il cherche B, il aura 2 recherches à faire. S'il cherche C, il aura 3 recherches à faire, etc.
- 3 - Utilisez le NEXT sans les noms de variables des instructions FOR.
NEXT est plus rapide que NEXT I car le B.E.V.F. ne vérifie pas alors si la variable spécifiée dans le NEXT correspond à la variable de la boucle FOR en cours d'exécution.
- 4 - Quand le B.E.V.F. rencontre en cours d'exécution une instruction telle que GOTO 1000, il scrute le programme à partir de la première ligne jusqu'à ce qu'il rencontre la ligne 1000.
C'est pourquoi, pour améliorer la vitesse, des lignes fréquemment appelées auront intérêt à être placées en début du programme.



Notes

ANNEXE F :

VALEURS DÉCIMALES DES MOTS DU LANGUAGE B.E.V.F.

<u>Valeur décimale</u>	<u>Mot</u>	<u>Valeur décimale</u>	<u>Mot</u>	<u>Valeur décimale</u>	<u>Mot</u>
128	END	159	FLASH	190	GET
129	FOR	160	COLOR=	191	NEW
130	NEXT	161	POP	192	TAB(C
131	DATA	162	VTAB	193	TO
132	INPUT	163	HIMEM:	194	FN
133	DEL	164	LOMEM:	195	SPC(C
134	DIM	165	ONERR	196	THEN
135	READ	166	RESUME	197	AT
136	GR	167	RECALL	198	NOT
137	TEXT	168	STORE	199	STEP
138	PR#	169	SPEED=	200	+
139	IN#	170	LET	201	-
140	CALL	171	GOTO	202	*
141	PLOT	172	RUN	203	/
142	HLIN	173	IF	204	^
143	VLIN	174	RESTORE	205	AND
144	HGR2	175	&	206	OR
145	HGR	176	GOSUB	207	>
146	HCOLOR=	177	RETURN	208	=
147	H PLOT	178	REM	209	<
148	DRAW	179	STOP	210	SGN
149	XDRAW	180	ON	211	INT
150	HTAB	181	WAIT	212	ABS
151	HOME	182	LOAD	213	USR
152	ROT=	183	SAVE	214	FRE
153	SCALE=	184	DEF	215	SCRNC
154	SHLOAD	185	POKE	216	PDL
155	TRACE	186	PRINT	217	POS
156	NOTRACE	187	CONT	218	SQR
157	NORMAL	188	LIST	219	RND
158	INVERSE	189	CLEAR	220	LOG

Valeur décimale

Mot

221	EXP
222	COS
223	SIN
224	TAN
225	ATN
226	PEEK
227	LEN
228	STR\$
229	VAL
230	ASC
231	CHR\$
232	LEFT\$
233	RIGHT\$
234	MID\$

ANNEXE G :

126

MOTS RÉSERVÉS DU B.E.V.F.

&

ABS	AND	ASC	AT	ATN			
CALL	CHR%	CLEAR	COLOR=	CONT	COS		
DATA	DEF	DEL	DIM	DRAW			
END	EXP						
FLASH	FN	FOR	FRE				
GET	GOSUB	GOTO	GR				
HCOLOR=	HGR	HGR2	HIMEM:	HLIN	HOME	HPLOT	HTAB
IF	IN#	INPUT	INT	INVERSE			
LEFT%	LEN	LET	LIST	LOAD	LOG	LOMEM:	
MID%							
NEW	NEXT	NORMAL	NOT	NOTRACE			
ON	ONERR	OR					
PDL	PEEK	PLOT	POKE	POP	POS	PRINT	PR#
READ	RECALL	REM	RESTORE	RESUME	RETURN	RIGHT%	
	RND	ROT=	RUN				
SAVE	SCALE=	SCRN(SGN	SHLOAD	SIN	SPC(
	SPEED=	SQR	STEP	STOP	STORE	STR%	
TAB(TAN	TEXT	THEN	TO	TRACE		
USR							
VAL	VLIN	VTAB					
WAIT							
XPLOT	XDRAW						

Le B.E.V.F. code les mots réservés, ce qui fait que chaque mot réservé n'occupe qu'un seul octet en mémoire, c'est-à-dire la valeur de son code. Tout autre caractère dans le programme utilise 1 octet en mémoire. Voir à l'annexe F la valeur décimale des codes.



Le signe & ne fait pas partie des commandes du B.E.V.F. Il est utilisé dans le travail intérieur de l'ordinateur.

Si vous l'utilisez comme une instruction, cela provoquera un saut inconditionnel à l'adresse mémoire 3F5. Faites **RESET** **CTRL** **RETURN** pour reprendre le contrôle de l'ordinateur.



XPLOT est un mot réservé mais ne correspond pas à une commande existante du B.E.V.F.

Certains des mots réservés ne sont reconnus par le B.E.V.F. que dans certaines conditions:

COLOR, HCOLOR, SCALE, SPEED, ROT
sont interprétés comme des mots réservés seulement si le premier caractère qui les suit (différent de l'espace) est le signe =

SCR, SPC, TAB
sont interprétés comme des mots réservés seulement si le premier caractère qui les suit (différent de l'espace) est une parenthèse ouvrante (.

HIMEM:, LOMEM:
doivent être suivis de deux points (:) pour être interprétés comme des mots réservés.

ATN
est interprété comme un mot réservé s'il n'y a pas d'espace entre T et N. Si un espace s'intercale entre T et N c'est le mot réservé AT qui est interprété au lieu de ATN.

TO
est considéré comme un mot réservé à moins qu'il n'y ait un espace entre le T et le O, dans ce cas si un A précède le TO c'est le mot réservé AT qui est interprété.

Pour éviter les inconvénients des mots réservés, vous pouvez utiliser des parenthèses:

```
100 FOR A = LOFT OR CAT TO 15
```

sera LISTÉ comme:

```
100 FOR A = LOF TO RC AT TO 15
```

mais:

```
100 FOR A = (LOFT) OR (CAT) TO 15
```

sera LISTÉ comme:

```
100 FOR A = (LOFT) OR (CAT) TO 15
```

Notes



ANNEXE H :

POUR CONVERTIR UN PROGRAMME " BASIC " EN B.E.V.F.

Bien que les BASIC utilisés sur différents ordinateurs aient de nombreux points communs, il y a certaines incompatibilités dont vous devez tenir compte pour traduire des programmes du BASIC en B.E.V.F.

- 1 - Les index des tableaux sont utilisés entre crochets ([]) dans certains BASIC, le B.E.V.F. les traite entre parenthèses ().
- 2 - Chaînes de caractères.
Certains BASIC vous obligent à déclarer la dimension d'une chaîne de caractères avant de l'employer. Supprimez toutes les instructions de ce type dans le programme pour le copier en B.E.V.F.
Dans ces BASIC, une déclaration telle que DIM A\$(I, J) attend un tableau de J chaînes où chaque chaîne a une longueur I.
Pour le B.E.V.F., DIM A\$(J) a la même fonction.
Pour la concaténation des chaînes, le B.E.V.F. utilise "+" et non ",", " ou "&"
Le B.E.V.F. utilise LEFT\$, RIGHT\$ et MID\$ pour extraire des parties d'une chaîne.
Certains BASIC ne possèdent pas ces fonctions mais utilisent A\$(I) pour accéder au Ième caractère de la chaîne et utilisent A\$(I, J) pour extraire une sous-chaîne de la Ième à la Jème position de A\$.

Convertissez comme suit:

<u>BASIC</u>	<u>B.E.V.F.</u>
A\$(I)	MID\$(A\$, I, 1) ou LEFT\$(A\$, I)
A\$(I, J)	MID\$(A\$, I, J-I + 1)

Cela suppose que l'instruction de la sous-chaîne était placée dans une expression ou à droite d'une assignation. Si la manipulation de la chaîne se fait à gauche de l'assignation, effectuez alors les conversions suivantes:

<u>BASIC</u>	<u>B.E.V.F.</u>
A\$(I)=X\$	A\$=LEFT\$(A\$, I-1) + X\$ + MID\$(A\$, I+1)
A\$(I, J)=X\$	A\$=LEFT\$(A\$, I-1) + X\$ + MID\$(A\$, J+1)

- 3 - Certains BASIC permettent des assignations multiples telles que:
500 B = C = Ø Instruction qui veut dire B et C à Ø.
Cela aurait eu un effet totalement différent en B.E.V.F. Tous les signes = après le premier signe = seraient considérés comme des opérations logiques.
B serait mis à -1 si C vaut Ø
B serait mis à Ø si C est différent de Ø.
Pour la conversion, il faut réécrire la ligne:
500 C = Ø : B = C
- 4 - Certains BASIC utilisent "/" au lieu de ":" comme séparateur d'instruction sur une même ligne. Convertissez chaque "/" en ":"

5 - Certains BASIC possèdent des fonctions MAT qui permettent le calcul matriciel, le B.E.V.F. ne possède pas ces fonctions. Il faut les recréer en utilisant les boucles FOR...NEXT.

Notes

ANNEXE I :

CARTE DE LA MÉMOIRE

<u>ZONE MEMOIRE</u>	<u>DESCRIPTION</u>
0.1FF	Espace de travail du programme, non disponible pour l'utilisateur.
200.2FF	Zone de stockage des caractères au clavier.
300.3FF	Disponible pour les petits programmes langage machine de l'utilisateur.
400.7FF	Image mémoire de la page 1 du texte et du graphisme basse résolution.
800.XXX	Avec la version carte ROM du B.E.V.F. programme de l'utilisateur et zones de stockage des variables. XXX est l'adresse maximum disponible en mémoire RAM
800.2FFF	Interpréteur de la version cassette du B.E.V.F.
2000.3FFF	B.E.V.F. carte ROM seulement, page 1 de la haute résolution.
3000.XXX	Version cassette du B.E.V.F., programme de l'utilisateur et zones de stockage des variables. XXX est l'adresse maximum disponible en mémoire RAM. XXX peut être diminuée en réservant de la place mémoire pour les programmes en langage machine ou pour mémoriser une page HR.
4000.5FFF	Page 2 de la haute résolution
C000.CFFF	Adresses hardware des E/S.
D000.DFFF	Expansions futures en ROM.
D000.F7FF	Interpréteur de la version carte ROM du B.E.V.F. (Interpréteur vers le haut).
E000.F7FF	BASIC Entier.
F800.FFFF	Moniteur de l'ITT 2020.

DIAGRAMME DE LA CARTE MÉMOIRE DU B.E.V.F.

132

VERSION CASSETTE

Pointeurs
Système de gestion du disque (si le
disque est utilisé)

§73 - §74 (HIMEM:)
Avant qu'il soit modifié par l'utilisa-
teur HIMEM: est automatiquement fixé à
l'adresse maximum disponible en mémoire
RAM

Chaînes
§ 6F - §70

Espace libre
Comprenant les registres des graphiques
haute résolution (seule la page 2 est
disponible en version cassette)
ATTENTION: l'espace réservé aux chaînes
peut déborder sur la page de haute réso-
lution ou sur des programmes en langage
machine.
Pour provoquer un "nettoyage interne" du
B.E.V.F. utilisez fréquemment X = FRE(0)
dans votre programme.

§6D - §6E
Pointeurs des tableaux et des tableaux
de chaînes
§6B - §6C

Variables
§69 - §6A (LOMEM:)

§AF - §B0
PROGRAMME
§67 - §68

B.E.V.F.

VERSION ROM

§3001

§2FFF

§801

§801

F7FF

D000



Notes

" PEEK, POKE, CALL " + TABLES DES VARIABLES ET TABLEAUX

Vous pouvez, à l'aide des PEEK, POKE et CALL profiter de certaines possibilités offertes par le B.E.V.F. Vous remarquerez que l'on retrouve parfois les commandes du B.E.V.F.

Les actions de sélection sont en général dépendantes d'adresses. Toute commande concernant cette adresse aura le même effet sur "l'interpréteur".

Par exemple:

POKE -16304,0

Vous auriez obtenu le même effet en plaçant à cette adresse n'importe quelle valeur entre 0 et 255 ou en exécutant un PEEK à cette adresse:

X = PEEK (-16304)

Ces adresses sont des adresses spéciales et ce procédé ne s'applique pas aux commandes où il faut placer une valeur bien définie dans l'adresse.
Par exemple, pour déplacer le curseur.

FIXER LA TAILLE DE LA FENÊTRE DE TEXTE

Il existe 4 commandes POKE pour modifier la taille de la fenêtre de texte sur l'écran. On peut modifier la marge à gauche, la longueur, la marge haute et la marge basse de la fenêtre de texte.

En modifiant la fenêtre de texte, vous n'effacez pas les caractères sur l'écran et vous ne placez pas le curseur à l'intérieur des nouvelles limites de la fenêtre.
(Utilisez HOME ou HTAB et VTAB pour cela).

La commande VTAB ignore la taille de la fenêtre et travaille donc par rapport aux bords de l'écran. Le texte au-dessus de la fenêtre s'affiche correctement, alors que le texte sous la fenêtre ne s'affiche que sur une ligne. HTAB peut aussi déplacer le texte hors de la fenêtre mais vous ne pourrez afficher qu'un seul caractère à l'extérieur.

Un changement sur la largeur de la fenêtre est immédiat, mais un changement sur la marge gauche n'est effectif que si le curseur est renvoyé contre la marge gauche.



Le texte affiché sur l'écran est l'image d'une partie de la mémoire (page 1 du texte).


L'écran TV reflète toujours la même partie de la mémoire et affiche ce que l'ITT 2020 a écrit dedans. Tout va bien tant que vous ne modifiez pas la fenêtre de texte. Mais si vous fixez la marge gauche à, par exemple, 255, alors que sa valeur maximum devrait être 40, vous demandez à l'ITT 2020 d'écrire au-delà de la zone mémoire réservée du texte. Vous risquez de détruire une partie de votre programme ou même le B.E.V.F. en version cassette.

Pour plus de sécurité, limitez les modifications de la fenêtre sur les 40 caractères, 24 lignes de l'écran.

POKE 32, G

Fixe la marge gauche de l'écran à la valeur spécifiée par G, entre 0 et 39 où 0 est la

position la plus à gauche. Ce changement ne sera effectif que lorsque le curseur sera revenu contre la marge gauche.

 La largeur de la fenêtre n'est pas changée par cette commande: vous avez donc à la modifier en la déplaçant de la même quantité que vous avez utilisé pour la marge gauche. Pour préserver le programme et le B.E.V.F. cassette; réduire d'abord la largeur avant de modifier la marge gauche.

POKE 33, L

Fixe la largeur de la fenêtre de texte à la valeur spécifiée par L entre 1 et 40.



Ne pas fixer L à 0, cela détruit le B.E.V.F. cassette.



Si L est inférieur à 33, l'instruction PRINT sur le troisième champ de tabulation risque d'afficher des caractères à l'extérieur de la fenêtre.

POKE 34, H

Fixe la marge haute de la fenêtre de texte à la valeur spécifiée par H, H doit être compris entre 0 et 23 et 0 est le haut de l'écran.

Ne descendez pas la marge haute plus bas que la marge basse (voir ci-dessous).

POKE 35, B

Fixe la marge basse de la fenêtre de texte à la valeur spécifiée par B, B doit être compris entre 0 et 24 où 24 est la dernière ligne de l'écran. Ne pas fixer la marge basse plus haute que la marge haute.

QUATRE COMMANDES RELATIVES AU TEXTE, A LA FENÊTRE DE TEXTE ET AU CLAVIER


CALL -936

Efface tous les caractères de la fenêtre de texte et place le curseur sur le coin en haut à gauche de la fenêtre de texte.

Effet identique à celui de la commande HOME et de la fonction du clavier  .

CALL -958

Efface tous les caractères (dans la fenêtre de texte) situés sous la ligne du curseur.

Même effet que .

Si le curseur est au-dessus de la fenêtre de texte, il efface les caractères de la marge gauche à la marge droite et jusqu'à la marge droite et jusqu'à la marge bas. Il n'y a aucun intérêt à utiliser cette commande avec le curseur placé sous la fenêtre de texte, en effet, seule la dernière ligne de la fenêtre est effacée.

CALL -868

Efface les caractères sur la ligne du curseur, du caractère suivant le curseur jusqu'à la marge droite. Même effet que .

CALL -922

Provoque un saut de ligne. Même effet que

CTRL
J

CALL -912

Remonte chaque ligne de texte (dans la fenêtre) d'une position. La ligne du haut est perdue, la seconde ligne devient la première, la ligne du bas devient vierge. Les caractères extérieurs à la fenêtre ne sont pas affectés.

X = PEEK (-16384)

Lit le clavier. Si X > 127 c'est qu'une touche a été enfoncée et X est la valeur ASCII de la touche avec le bit 7 à 1.

C'est utile dans les longs programmes quand l'ordinateur vérifie si l'utilisateur veut interrompre le programme pour communiquer de nouvelles données sans arrêter l'exécution du programme.

POKE -16368, 0

Réinitialise le strobe du clavier (indicateur à 1 si le clavier a été touché; à 0 sinon) pour que le prochain caractère puisse être lu.

Cette commande doit être effectuée immédiatement après avoir lu le clavier.

COMMANDES RELATIVES AU CURSEUR

CH = PEEK (36)

Met dans CH la position horizontale du curseur.

CH sera compris entre 0 et 39 et représente la position relative du curseur par rapport à la marge gauche de la fenêtre de texte, précédemment fixée par POKE 32, L

Donc, si vous fixez la marge gauche de la fenêtre par POKE 32, 5 le caractère le plus à gauche de la nouvelle fenêtre sera sur la 6ème colonne par rapport au bord gauche de l'écran. Si PEEK (36) donne une valeur de 5 c'est que le curseur est placé sur la 11ème colonne de l'écran.

Identique à la fonction POS (X)

POKE 36, CH

Déplace le curseur sur la CH + 1ème position horizontale par rapport à la marge gauche de la fenêtre de texte. (Exemple: POKE 36, 0 fera que le prochain caractère affiché sera contre la marge gauche).

Si la marge gauche de la fenêtre est fixée à 6 (POKE 32, 6) et que vous vouliez afficher un caractère sur la 3ème colonne par rapport au bord de l'écran, la marge gauche doit être modifiée avant que l'affichage (PRINT) se fasse. CH doit être plus petit ou égal à la largeur de la fenêtre de texte fixée par POKE 33, L mais CH doit être supérieur à 0. Comme HTAB, cette commande peut déplacer le curseur hors de la fenêtre de texte, mais vous ne pourrez alors afficher qu'un seul caractère.

CV = PEEK (37)

Lit la position verticale du curseur et l'assigne dans CV. CV est une position absolue par rapport au bord de l'écran.

CV = 0, le curseur est sur la première ligne

CV = 23 le curseur est sur la dernière ligne.

POKE 37, CV

Déplace le curseur verticalement et par rapport aux bords de l'écran. CV = 0 est la 1ère ligne, CV = 23 est la dernière.

COMMANDES RELATIVES AU GRAPHISME

Pour pouvoir afficher du texte et du graphisme, la mémoire de l'ITT 2020 est divisée en 4 zones d'affichage: les pages 1 et 2 du texte et les pages 1 et 2 de la haute résolution.

A - La page 1 du texte sert pour l'affichage des caractères et des graphiques couleur large résolution.

B - La page 2 du texte est située juste au-dessus de la page 1 du texte. Elle n'est pas facilement accessible à l'utilisateur. Comme en page 1, les informations de la page 2 peuvent être des caractères ou du graphisme couleur large résolution.

C - La page 1 haute résolution réside dans la mémoire de l'ITT 2020 entre le 8ème et le 16ème K. On sélectionne cette page par HGR. Si le mode mixte (texte + graphisme) est utilisé, les caractères sont pris dans la page 1 de texte.

D - La page 2 haute résolution réside dans la mémoire de l'ITT 2020 entre le 16ème et le 24ème K. On sélectionne cette page par HGR2. Si le mode mixte est utilisé, les caractères sont pris dans la page 2 du texte.

Pour utiliser les modes texte ou graphisme, vous pouvez utiliser les commandes du B.E.V.F. prévues à cet effet ou vous pouvez directement opérer sur les 4 "flags". Comme vu précédemment un PEEK ou un POKE sur une adresse positionne le "flag" d'une certaine manière et un PEEK ou un POKE sur une autre adresse positionne le "flag" d'une autre manière.

En résumé, les 4 "flags" peuvent sélectionner:

1 - Affichage de texte	(POKE -16303, 0)
Affichage de graphisme haute ou large résolution	(POKE -16304, 0)
2 - Page 1 de texte ou haute résolution	(POKE -16300, 0)
Page 2 de texte ou haute résolution	(POKE -16299, 0)
3 - Pages 1 et 2 de texte pour graphisme	(POKE -16298, 0)
Pages 1 et 2 de haute résolution	(POKE -16297, 0)
4 - Graphisme total haute et large résolution	(POKE -16302, 0)
Graphisme mixte haute et large résolution	(POKE -16301, 0)

POKE -16304, 0

Sélectionne le mode graphique couleur, sans nettoyer l'écran. Selon l'état des 3 autres "flags", le mode graphique sélectionné peut être de la large ou haute résolution de la page 1 ou 2, ou être du graphisme mixte ou total.

De même les commandes B.E.V.F.: GR sélectionne la page 1 ou 2 large résolution, mixte le texte et le graphisme et nettoie l'écran.

HGR sélectionne la page 1 de la haute résolution en mode mixte (graphisme + texte) et nettoie l'écran.

HGR2 sélectionne la page 2 de la haute résolution en mode graphisme total et nettoie l'écran.

POKE -16303, 0

Sélectionne le mode texte de l'affichage sans réinitialiser les dimensions de la fenêtre de texte. La page sélectionnée peut être la page 1 ou la page 2.

De même la commande du B.E.V.F. TEXT sélectionne le mode TEXTE de la page 1, réinitialise la fenêtre de texte sur les bords de l'écran et place le curseur en bas à gauche.

POKE -16302, 0

Sélectionne le mode graphisme total.

Selon l'état des autres "flags", la sélection peut se faire sur le mode texte, sur le mode large résolution 40 X 48 ou sur le mode haute résolution 360 X 192.

POKE -16301, 0

Sélectionne le mode graphique mixte, en faisant apparaître 4 lignes de texte en bas de l'écran.

Selon l'état des autres "flags", la portion du haut de l'écran peut être du texte, du graphisme large résolution 40 X 40 ou de la haute résolution.

Les deux portions de l'écran peuvent venir de la page 1 ou de la page 2.

POKE -16300, 0

Passe de la page 2 à la page 1, sans nettoyer l'écran ni bouger le curseur.

Nécessaire quand vous passez du B.E.V.F. au BASIC Entier, sinon c'est la page 2 qui s'affichera.

Selon l'état des autres "flags", l'affichage passe de la page 2 de texte à la page de texte 1, ou de la page 2 graphisme large résolution, ou de la page 1 graphisme haute résolution.

POKE -16299, 0

Passe de la page 1 à la page 2, sans nettoyer l'écran, ni bouger le curseur.

Selon l'état des autres "flags", l'affichage passe de la page texte 1 à la page texte 2 ou de la page graphisme large résolution 1 à la page graphisme large résolution 2, ou de la page 1 graphisme haute résolution à la page 2 graphisme haute résolution.

POKE -16298, 0

Passe de la page graphique haute résolution à la page graphique large résolution de même numéro (1 → 1, 2 → 2) sans nettoyer l'écran. Indispensable quand vous passez du B.E.V.F. au BASIC Entier.

Autrement l'instruction GR du BASIC Entier risque de vous afficher une page haute résolution.

Selon l'état des autres "flags", l'affichage peut passer de la page 1 graphisme haute résolution à la page 1 graphisme large résolution, ou de la page 2 graphisme haute résolution à la page 2 graphisme large résolution (en mode texte l'affichage n'est pas changé).

CALL -1994

Affiche sur les 20 premières lignes de l'écran un signe @ (page 1). Si vous êtes en mode graphisme large résolution, cette commande nettoie les 40 premières lignes du graphisme. Pas d'effet sur la page 2 du texte ou sur les pages haute résolution.

CALL -1998

Affiche des signes @ sur tout l'écran (page 1). Si vous êtes en page 1 du graphisme large résolution, l'écran est entièrement nettoyé. Pas d'effet sur les pages de haute résolution.

CALL 62450

Nettoie l'écran haute résolution de la page utilisée.

CALL 62454

"Paint" l'écran haute résolution dans la couleur (HCOLOR) du dernier point dessiné. Cette commande doit être précédée d'un HPLOT.

INSTRUCTIONS RELATIVES AUX LEVIERS DE COMMANDE ET AU HAUT-PARLEUR

X = PEEK (-16336)

Active le haut-parleur: produit un "clic" dans le haut-parleur.

X = PEEK (-16352)

Active la sortie cassette: produit un "clic" sur une cassette.

X = PEEK (-16287)

Lit le bouton du levier de commandes \emptyset . Si $X > 127$, le bouton a été pressé.

X = PEEK (-16286)

Même commande que ci-dessus mais pour le levier de commandes 1.

POKE -16296, 1

Met la sortie # \emptyset (connecteur jeu, broche 15) de la commande jeu "annonciateur" à l'état TTL haut en connecteur ouvert (3,5 V). C'est la condition arrêt.

POKE -16295, \emptyset

Met la sortie # \emptyset à l'état bas, ($\emptyset,3$ V). C'est la condition marche (courant maximal: 1,6 mA).

POKE -16294, 1

Met la sortie # 1 (connecteur jeu, broche 14) à l'état haut (3,5 V).

POKE -16293, \emptyset

Met la sortie # 1 à l'état TTL bas ($\emptyset,3$ V).

POKE -16292, 1

Met la sortie # 2 (connecteur jeu, broche 13) à l'état TTL haut (3,5 V).

POKE -16291, \emptyset

Met la sortie # 2 à l'état bas ($\emptyset,3$ V).

COMMANDES RELATIVES AUX ERREURS

X = PEEK (218) + PEEK (215) * 256

Cette instruction place dans X le numéro de ligne ou une erreur s'est déclenchée si l'instruction ONERR GOTO a été utilisée.

IF PEEK (216) > 127 THEN GOTO...

Si le bit 7 de la case mémoire 222 (indicateur d'erreur) est à 1, c'est qu'une instruction ONERR GOTO a été rencontrée.

POKE 216, 0

Réinitialise l'indicateur d'erreur, les messages d'erreurs se provoqueront alors normalement.

Y = PEEK (222)

Place dans la variable Y un code représentant le type d'erreur qui causa l'exécution d'une instruction ONERR GOTO. Voici la liste des erreurs:

<u>VALEUR DE Y</u>	<u>TYPE D'ERREUR</u>		
0	NEXT sans FOR		
16	SYNTAXE		
22	RETURN sans GOSUB		
42	plus de données		
53	quantité illégale		
69	dépassement de capacité		
77	plus de mémoire		
90	instruction non définie		
107	erreur d'index		
120	tableau redimensionné		
133	division par zéro		
163	erreur de type		
176	chaîne trop longue		
191	formule trop complexe		
224	fonction non définie		
254	mauvaise réponse à une instruction INPUT		
255	essai d'interruption par un <table border="1" style="display: inline-table; vertical-align: middle;"><tr><td>CTRL</td></tr><tr><td>C</td></tr></table>	CTRL	C
CTRL			
C			

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

Construit un sous-programme en langage machine à partir de la case mémoire 768, qui peut être utilisé comme un sous-programme de manipulation des erreurs.

Evitez certains inconvénients dus à l'utilisation de ONERR GOTO (pour PRINT et ?OUT OF MEMORY ERROR).

Utilisez la commande CALL 768 pour appeler le sous-programme.

VARIABLES ET TABLEAUX EN B.E.V.F.

VARIABLES

Pointeurs

§69-§6A

Réelles	Entières	Pointeurs de chaînes
NOM (+) 1er octet (+) 2ème octet	NOM (-) 1er octet (-) 2ème octet	NOM (-) 1er octet (+) 2ème octet
exposant 1er octet mantisse octet, le + significatif mantisse mantisse mantisse octet, le - significatif	octet poids fort octet poids faible ∅ ∅ ∅	longueur 1er octet adresse poids faible adresse poids fort ∅ ∅

TABLEAUX

Pointeurs

§6B-§6C

Réels	Entiers	Pointeurs de chaînes
NOM (+) 1er octet (+) 2ème octet	NOM (-) 1er octet (-) 2ème octet	NOM (-) 1er octet (+) 2ème octet
pointeur à la variable suivante octet poids faible octet poids fort	pointeur à la variable suivante octet poids faible octet poids fort	pointeur à la variable suivante octet poids faible octet poids fort
NOMBRE DE DIMENSIONS 1 octet	NOMBRE DE DIMENSIONS 1 octet	NOMBRE DE DIMENSIONS 1 octet
TAILLE DE LA Nième DIMENSION octet poids fort octet poids faible	TAILLE DE LA Nième DIMENSION octet poids fort octet poids faible	TAILLE DE LA Nième DIMENSION octet poids fort octet poids faible
TAILLE DE LA 1ère DIMENSION octet poids fort octet poids faible	TAILLE DE LA 1ère DIMENSION octet poids fort octet poids faible	TAILLE DE LA 1ère DIMENSION octet poids fort octet poids faible

REELLE ($\emptyset, \emptyset \dots \emptyset$) exposant 1 octet mantisse octet le +significatif mantisse mantisse mantisse octet le - significatif	ENTIER % ($\emptyset, \emptyset \dots \emptyset$) octet poids fort octet poids faible	CHAINE% ($\emptyset, \emptyset, \emptyset \dots \emptyset$) longueur 1 octet adresse poids faible adresse poids fort
	ENTIER % (N,N...N) octet poids fort octet poids faible	
RELLE (N,N,N...N) exposant 1 octet mantisse octet le + significatif mantisse mantisse mantisse octet le - significatif		

§6D-§6E

Les signes (+) et (-) indiquent la valeur du bit 7 des octets réservés au nom.
 +: le bit 7 est à 1
 -: le bit 7 est à \emptyset .
 Les chaînes sont stockées dans leur ordre d'entrée à partir de HIMEM: et en descendant.
 La table des chaînes pointe sur le 1er caractère de la chaîne. Si les chaînes sont modifiées, les pointeurs sont changés, quand toute la mémoire disponible est utilisée, le "nettoyage interne" supprime alors les chaînes abandonnées.
 (Le "nettoyage interne" se fait par FRE (X)).

Tous les tableaux sont stockés avec l'index le plus à droite s'incrémentant le plus lentement. C'est-à-dire que les nombres dans le tableau A% avec A% (\emptyset, \emptyset) = \emptyset , A% (1, \emptyset) = 1, A% ($\emptyset, 1$) = 2, A% (1,1) = 3 seront stockées en mémoire dans le même ordre.



Notes

ANNEXE K :

Code des touches par
PFPR (-16384)

CODES " ASCII " DES CARACTÈRES

DEC : code ASCII décimal
HEX : code ASCII hexadécimal
CAR : nom ASCII du caractère
n/a : non accessible à partir du clavier de l'ITT 2020

Code des touches par
PFPR (-16384)

DEC	HEX	CAR	QUE TAPER
0	00	NULL	CTRL @
1	01	SOH	CTRL A
2	02	STX	CTRL B
3	03	ETX	CTRL C
4	04	ET	CTRL D
5	05	ENQ	CTRL E
6	06	ACK	CTRL F
7	07	BEL	CTRL G
8	08	BS	CTRL H ou ←
9	09	HT	CTRL I
10	0A	LF	CTRL J
11	0B	VT	CTRL K
12	0C	FF	CTRL L
13	0D	CR	CTRL M ou RETURN
14	0E	SO	CTRL N
15	0F	SI	CTRL O
16	10	DLE	CTRL P
17	11	DC1	CTRL Q
18	12	DC2	CTRL R
19	13	DC3	CTRL S
20	14	DC4	CTRL T
21	15	NAK	CTRL U ou →
22	16	SYN	CTRL V
23	17	ATB	CTRL W
24	18	CAN	CTRL X
25	19	EM	CTRL Y
26	1A	SUB	CTRL Z

27	1B	ESCAPE	ESC			
28	1C	FS	n/a			
29	1D	GS	CTRL	SHIFT	M	
30	1E	RS	CTRL	SHIFT	N	
31	1F	US	n/a			
32	20	SPACE	SPACE			
33	21	!	!			
34	22	"	"			
35	23	#	#			
36	24	\$	\$			
37	25	%	%			
38	26	&	&			
39	27	'	'			
40	28	((
41	29))			
42	2A	*	*			
43	2B	+	+			
44	2C	,	,			
45	2D	-	-			
46	2E	.	.			
47	2F	/	/			
48	30	0	0			
49	31	1	1			
50	32	2	2			
51	33	3	3			
52	34	4	4			
53	35	5	5			
54	36	6	6			
55	37	7	7			
56	38	8	8			
57	39	9	9			
58	3A	:	:			
59	3B	;	;			
60	3C	<	<			
61	3D	=	=			

62	3E	>	>
63	3F	?	?
64	40	@	@
65	41	A	A
66	42	B	B
67	43	C	C
68	44	D	D
69	45	E	E
70	46	F	F
71	47	G	G
72	48	H	H
73	49	I	I
74	4A	J	J
75	4B	K	K
76	4C	L	L
77	4D	M	M
78	4E	N	N
79	4F	O	O
80	50	P	P
81	51	Q	Q
82	52	R	R
83	53	S	S
84	54	T	T
85	55	U	U
86	56	V	V
87	57	W	W
88	58	X	X
89	59	Y	Y
90	5A	Z	Z
91	5B	[<i>alt</i>
92	5C	\	<i>alt</i>
93	5D]]
94	5E	^	^
95	5F	_	<i>alt</i>

SHIFT M



Notes

Notes





Notes

ANNEXE L :

L'ORGANISATION DE LA PAGE ZÉRO AVEC B.E.V.F.

ADRESSE EN HEXADÉCIMAL	UTILISATION
\$0 - \$5 $\emptyset - 5$	Instructions de saut pour continuer en B.E.V.F. (RESET \emptyset RETURN est équivalent au RESET CTRL RETURN ^{3D0G} du BASIC Entier).
\$A - \$C 10 - 12	Emplacement pour l'instruction de saut de la fonction USR (voir USR).
\$D - \$17 13 - 23	Compteurs et indicateurs utilisés par le B.E.V.F.
\$20 - \$4F 32 - 79	Emplacements réservés au moniteur du B.E.V.F.
\$50 - \$61 80 - 97	Pointeurs utilisés par le B.E.V.F.
\$62 - \$66 98 - 102	Résultat de la dernière multiplication ou division.
\$67 - \$68 103 - 104	Pointeur de début de programme. Initialement fixé à \$801 sur la version carte ROM et à \$3001 sur la version cassette.
\$69 - \$6A 105 - 106	Pointeur de début de table des variables. Indique aussi la fin du programme. Plus 1 ou 2 modifiables par LOMEM.
\$6B - \$6C 107 - 108	Pointeur de début de la table des tableaux.
\$6D - \$6E 109 - 110	Pointeur de fin de la zone de stockage numérique (fin des tables).
\$6F - \$70 111 - 112	Pointeur du début de stockage des chaînes de caractères. Les chaînes sont stockées du pointeur à la fin de la mémoire.
\$71 - \$72 113 - 114	Pointeur général.
\$73 - \$74 115 - 116	Adresse la plus haute disponible avec le B.E.V.F. + 1. Initialement fixée à la dernière adresse RAM disponible.
\$75 - \$76 117 - 118	Ligne de programme en cours d'exécution.
\$77 - \$78 119 - 120	Ligne de la dernière instruction du programme exécuté avant interruption (END, STOP, CTRL C).
\$79 - \$7A 121 - 122	Pointeur de l'adresse de l'instruction à exécuter.

§7D - §7C	125 - 124	Numéro de ligne de la donnée (DATA) étant lue (READ).
§7D - §7E	125 - 126	Pointeur de l'adresse mémoire de la donnée (DATA) étant lue (READ).
§7F - §80	127 - 128	Pointeur sur la nature d'un INPUT. Fixé à §201 pendant une instruction INPUT. Est fixé à la donnée (DATA) lue pendant une instruction READ.
§81 - §82	129 - 130	Conserve le nom de la dernière variable utilisée.
§83 - §84	131 - 132	Pointeur sur la valeur de la dernière variable utilisée.
§85 - §9G	133 - 156	Usage général (pour le B.E.V.F.).
§9D - §A3	157 - 163	Accumulateur virgule flottante.
§A4	164	Usage général pour les séquences mathématiques en virgule flottante.
§A5 - §AB	165 - 171	Deuxième accumulateur virgule flottante.
§AC - §AE	172 - 174	Usage général pointeurs/indicateurs (utilisés par le B.E.V.F.).
§AF - §B0	175 - 176	Pointeur de fin de programme (non modifié par LOMEM:).
§B1 - §B8	177 - 184	Sous-programme d'introduction des caractères. Le B.E.V.F. se branche ici chaque fois qu'il a besoin d'un caractère.
§B8 - §B9	184 - 185	Pointeur sur le dernier caractère communiqué au B.E.V.F.
§C9 - §CD	201 - 205	Nombre aléatoire
§D0 - §D5	208 - 213	Pointeurs de haute résolution.
§D8 - §DF	216 - 223	Pointeurs de ONERR.
§E0 - §E2	224 - 226	Coordonnées X et Y en haute résolution.
§E4	228	Octet couleur en haute résolution.
§E5 - §E7	229 - 231	Usage général pour la haute résolution.
§E8 - §E8	232	Pointeur du début de table de construction des figures.
§EA	234	Compteur de collision pour la haute résolution.

§F0 - §F3 240 - 243

Indications d'usage général.

152

§F4 - §F8 244 - 248

Pointeurs ONERR.

Notes

ANNEXE M : DIFFÉRENCES ENTRE B.E.V.F. ET " BASIC ENTIER "

DIFFÉRENCES ENTRE LES COMMANDES

Voici la liste des commandes existant en B.E.V.F. mais pas en BASIC Entier:

ATN						
CHR\$	COS					
DATA	DEF FN	DRAW				
EXP						
FLASH	FN	FRE				
GET						
HCOLOR	HGR	HGR2	HIMEM:	HOME	HLOT	
INT	INVERSE					
LEFT\$	LOG	LOMEM:				
MID\$						
NORMAL						
ON...GOSUB		ON...GOTO		ONERR GOTO		
POS						
READ	RECALL	RESTORE	RESUME	RIGHT\$	ROT	
SCALE	SHLOAD	SIN	SPC	SPEED	SQR	STOP
	STORE	STR\$				
TAN						
USR						
VAL						
WAIT						
XDRAW						

Voici la liste des commandes existant en BASIC Entier mais pas en B.E.V.F.:

AUTO	
DSP	
MAN	MOD

Voici la liste des commandes d'action similaire mais de nom différent:

BASIC ENTIER	B.E.V.F.
CLR	CLEAR
CON	CONT
TAB	HTAB (remarque: le B.E.V.F. accepte aussi TAB).
GOTO X * 10 + 100	ON X GOTO 100, 110, 120
GOSUB X * 100 + 1000	ON X GOSUB 1000, 1100, 1200
CALL -936	HOME (ou CALL -936)
POKE 50,127 = CALL-384	INVERSE
POKE 50,255 = CALL-380	NORMAL
X	X% (%indique une variable entière).
#	< OU >

AUTRES DIFFÉRENCES

En BASIC Entier, la validité d'une instruction est vérifiée quand vous appuyez sur la touche **RETURN**, en B.E.V.F. une telle vérification se fait à l'exécution du programme.

GOTO et GOSUB doivent être suivis d'un numéro de ligne en B.E.V.F., en BASIC Entier les expressions arithmétiques sont autorisées.

Les variables ou constantes réelles sont utilisables en B.E.V.F. mais pas en BASIC Entier.

Seuls les deux premiers caractères d'un nom de variable sont reconnus par le B.E.V.F., en BASIC Entier le nom entier de la variable est retenu.

Les opérations sur les chaînes se définissent différemment en B.E.V.F. et en BASIC Entier. En BASIC Entier, les tableaux de chaînes de caractères n'existent pas et les variables alphanumériques doivent être dimensionnées.

Les tableaux peuvent être à plusieurs dimensions en B.E.V.F., ils ne peuvent l'être que d'une seule en BASIC Entier.

Le B.E.V.F. met automatiquement les tableaux à 0 avec RUN ou un CLEAR, l'utilisateur doit le faire lui-même en BASIC Entier.

Quand en BASIC Entier l'assertion IF...THEN est fausse, seule la portion de l'instruction suivant le THEN est ignorée. Quand l'assertion est fausse en B.E.V.F. c'est tout le reste de la ligne de programme qui est ignoré et l'exécution continue alors à la ligne de programme suivante.

En B.E.V.F. l'instruction TRACE affiche le numéro de ligne de chaque instruction sur une ligne utilisant des instructions multiples. En BASIC Entier, seul le numéro de ligne de la première instruction s'affiche.

En B.E.V.F. les POKE, PEEK et CALL doivent être compris entre 0 et 65535.

En BASIC Entier, les adresses supérieures à 32767 doivent être complétées à deux (32769 devient -32767, 32768 peut être appelé par -32767-1).

END est facultatif en B.E.V.F. mais obligatoire en BASIC Entier.

NEXT doit être suivi d'un nom de variable en BASIC Entier, c'est facultatif en B.E.V.F.

En BASIC Entier, la syntaxe de l'INPUT est:

```
INPUT [chaîne,] {var,}
```

Si var est un vara, l'INPUT affiche un ? avec ou sans la chaîne facultative. Si var est un varc, le point d'interrogation ne s'affiche pas.

En B.E.V.F. la syntaxe de l'INPUT est:

```
INPUT [chaîne;] {var,}
```

Si la chaîne facultative est omise, le B.E.V.F. affiche un ? sinon seule la chaîne s'affiche.

GLOSSAIRE ALPHABÉTIQUE DES DÉFINITIONS SYNTAXIQUES ET ABRÉVIATIONS

Référez-vous au chapitre 2 pour une présentation logique de ces définitions.
Le symbole := veut dire "est au moins partiellement défini comme".

caractère := lettre|chiffre|spéciaux

caractère
alphanumérique := lettre|chiffre

caractère de
reconnaissance du B.E.V.F. :=]

chaîne := "{caractère}"
Une chaîne occupe 1 octet (8 bits) pour sa longueur, 2 octets pour son pointeur et 1 octet pour chaque caractère.
:= "{caractère}" RETURN
Ce type de chaîne ne peut apparaître qu'en fin de ligne.

chaîne de
caractères
alphanumériques := chaîne

chaîne nulle := ""

chiffre := 1|2|3|4|5|6|7|8|9|ø

CTRL := Tenir enfoncée la touche CTRL pendant qu'une autre touche est enfoncée.

dimension := (extra { }, extra { | })
La dimension maximale est de 89 mais il sera limité en pratique par la taille mémoire disponible. Extra doit être positive et convertie en entier.

entier := [+/-] { chiffre }
Les entiers doivent varier entre -32767 et 32767. Lors de la conversion réel à entier, le B.E.V.F. applique en fait la fonction partie entière (INT) le nombre réel est alors arrondi par défaut à sa valeur entière. Néanmoins ce n'est pas vrai pour des nombres approchant de vraiment très près la valeur de l'entier supérieur.
Par exemple:
A% = 123.999 999 959 999
PRINT A%
123
A% = 123.999 999 96
PRINT A%
124

nom de variable entière := name
 Un réel peut être stocké dans une variable entière, mais B.E.V.F. convertit d'abord ce réel en entier.

nom de variable réelle := nom

nom de variable de chaîne := nom \$

numéro de ligne := numligne

numligne := {chiffre}
 Les numéros des lignes doivent être compris entre 0 et 63999 sinon le message ?SYNTAX ERROR (erreur de syntaxe) apparaîtra.

op := opal | opa

opa := + | - | * | / | ^

opal := AND | OR | = | > | < | <> | >< | >= | => | <= | =<
 NOT est volontairement absent de cette liste.

opc := +

opérateur := op

opérateur arithmétique := opa

opérateur arithmétique logique := opal

opérateur de chaîne := opc

opérateur logique de chaîne := oploc

oploc := = | > | >= | => | < | <= | =< | <> | ><

réel := [+|-] {chiffre} [.{chiffre}] [E +|-] chiffre [chiffre]
 := [+|-] [{chiffre}].[{chiffre}] [E+|-] chiffre [chiffre]
 La lettre E des nombres réels indique la présence d'un exposant (E est une abréviation de $\times 10^{\text{E}}$) le nombre suivant E est la puissance. Avec B.E.V.F. les réels doivent être compris entre -1E38 et 1E38 sinon le message ?OVERFLOW ERROR de dépassement de capacité s'affichera. En utilisant les additions et les soustractions il est possible de générer des nombres aussi grands que 1.7E38 sans que le message d'erreur s'affiche.
 Un réel dont la valeur absolue est plus petite que 2.9388E -39 vaudra

séparateur := ~|(')|=|-|+|Λ|>|<|/|*|,|;|:
 Un nom ne doit pas être séparé d'un mot réservé qui le précède ou le suit par un de ces délimiteurs.

spéciaux := !|#|§|%|&|'|(|)|*|:|=|-|_|+|;|?|/|>|.|<|,|]|Λ|"
 Les caractères de contrôle (caractères devant être tapés en tenant enfoncée la touche **CTRL**) et les caractères sans effet sont aussi des spéciaux. B.E.V.F. utilise le crochet droit (]) simplement comme caractère de reconnaissance du langage, il est utilisé dans ces pages comme un métasymbole.

symboles spéciaux
 utilisés par le B.E.V.F. := spéciaux

var := vara|varc

vara := nom|nom#
 Toutes les variables occupent 7 octets en mémoire: 2 pour le nom, 5 pour la valeur réelle ou entière.
 := vara dimension

varc := nom § nom § dimension
 Le pointeur de chaîne et le nom de la variable occupent tous deux 2 octets en mémoire. La longueur de la chaîne et chaque caractère la composant occupent 1 octet en mémoire.

variable := var

variable arithmétique := vara

variable de chaîne := varc
 := variable alphanumérique.



Notes

ANNEXE O :

162

RÉSUMÉ DES COMMANDES DU B.E.V.F.

La dernière page de ce manuel contient un index alphabétique des commandes et vous donne les pages où les commandes sont expliquées en détail.

ABS (-3.451)

Donne la valeur absolue de l'argument. Ici 3.451

ASC ("QUI")

Donne la valeur du code ASCII du premier caractère de l'argument. Ici 81 (ASCII de Q).

ATN (2)

Donne l'arc tangente en radians de l'argument. Ici 1.10714872

CALL -922

Provoque l'exécution d'un sous-programme en langage machine à l'adresse mémoire (en décimal) spécifié. L'exemple ici provoquera un saut de ligne.

CHR\$ (65)

Donne le caractère correspondant à la valeur de l'argument, qui doit être entre 0 et 255. Ici A.

CLEAR

Met les variables, les tableaux et les chaînes à zéro.

COLOR = 12

Sélectionne la couleur pour les dessins de large résolution. Dans cet exemple, la couleur est: rouge clair. COLOR est mis à 0 par GR.

Voici la liste des couleurs et de leurs codes:

0	:	noir	8	:	marron
1	:	bleu outremer	9	:	bleu clair
2	:	vert bouteille	10	:	vert pomme
3	:	bleu océan	11	:	turquoise
4	:	rouge foncé	12	:	rouge clair
5	:	violet	13	:	vieux rose
6	:	ocre	14	:	jaune
7	:	mauve	15	:	blanc

Pour déterminer la couleur d'un point, utilisez la fonction SCRN.

CONT

Si le programme a été interrompu par CTRL, CONT recommence l'exécution du programme à

l'instruction suivant la dernière instruction exécutée (comme par exemple GOSUB). L'exécution ne recommence pas forcément à la ligne suivante. Rien n'est effacé. Après RESET ØG RETURN, la CONTinuation du programme peut être refusée par l'ordinateur, certains pointeurs ou certaines piles ayant été réinitialisés.

CONT ne marchera pas si vous avez:

- 1 - modifié, ajouté ou supprimé une ligne de programme.
- 2 - provoqué un message d'erreur après l'arrêt d'exécution.

COS (2)

Calcule le cosinus de l'argument, qui doit être en radians. Dans cet exemple l'ordinateur retourne: -.415 146 836

CTRL
C

Utilisé pour interrompre un programme en cours d'exécution ou un listing. On peut l'utiliser aussi pour interrompre un INPUT si CTRL est le premier caractère introduit.

L'INPUT n'est pas interrompu tant que RETURN n'a pas été tapé.

CTRL
X

Indique à l'ITT 2020 d'ignorer la ligne que vous venez de taper. Aucune ligne n'est modifiée ou supprimée. Le caractère \ s'affiche à la fin de la ligne ignorée.

DATA JEAN "CODE 32" , 25.45, -6

Crée une liste d'éléments pouvant être exploitée par l'instruction READ. Dans cet exemple le premier élément est le littéral JEAN, le deuxième la chaîne "CODE 32", le troisième le nombre réel 23.45 et le dernier l'entier -6

DEF FN A(W) = 2 * W + W

Permet à l'utilisateur de définir des fonctions (sur une ligne BASIC). La fonction doit être préalablement définie en utilisant DEF, ce n'est qu'après cette définition qu'elle peut être utilisée dans le programme. L'exemple montre comment définir une fonction FN A(W), pouvant être utilisée ultérieurement dans le programme sous la forme, par exemple de: FN A(25) ou bien FN (Q * Q) etc.

FN A(23) assignera à W la valeur 23 dans le calcul de $2 * W + W$ et la fonction sera évaluée à $2 * 23 + 23$, c'est-à-dire 69. Si Q vaut 2, le B.E.V.F. assignera à W la valeur $2 * 2$, c'est-à-dire 4 donc FN A(4) sera calculé et le résultat est $4 * 2 + 4$, 12

DEL 26, 53

Détruit du programme toutes les lignes comprises entre les deux numéros. Dans l'exemple, toutes les lignes entre 26 et 53 (inclus) disparaissent. Pour détruire une ligne, par exemple la 350, utilisez la forme DEL 350,350 ou plus simplement tapez 350 puis RETURN.

DIM AGE (20, 3) , NOM\$ (50)

L'instruction DIM réserve de la place mémoire pour des tableaux spécifiés, avec index variant entre 0 et les nombres spécifiés dans la déclaration DIM. Dans l'exemple, le tableau AGE (20, 3) aura $(20 + 1) * (3 + 1)$, 84 éléments de nombres réels. NOM\$ réservera la place pour $50 + 1$, donc 51 chaînes alphanumériques de toutes longueurs. Si un tableau est utilisé avant d'avoir été déclaré par DIM, le B.E.V.F. réserve automatiquement 10 éléments pour chaque dimension. Les tableaux sont mis à zéro quand RUN ou CLEAR est exécuté.

DRAW 4 AT 50, 100

Dessine la 4ème figure de la table de construction des figures, en haute résolution, centrée sur X = 50 et Y = 100. La couleur, la rotation et l'échelle doivent avoir été préalablement définies avant d'utiliser DRAW.

END

Provoque l'arrêt d'exécution d'un programme, rend le contrôle de l'ordinateur à l'utilisateur. Aucun message ne s'affiche.

ESC	ESC	ESC	ESC
A	B	C	D

La touche **ESC** peut être utilisée en conjonction avec les autres touches A, B, C ou D pour déplacer le curseur sans affecter les caractères traversés. Pour déplacer le curseur d'une unité: pressez d'abord la touche **ESC** puis, après l'avoir relâchée, la lettre choisie.

COMMANDE DEPLACE LE CURSEUR D'UN ESPACE

ESC	à droite
A	
ESC	à gauche
B	
ESC	en bas
C	
ESC	en haut
D	

EXP (2)

Calcule l'exponentielle de l'argument. ($e = 2.718289$).
Dans l'exemple 7.3890561 sera renvoyé.

FLASH

Sélectionne le mode d'affichage clignotant de l'ordinateur (seulement pour les caractères affichés par l'ordinateur), les caractères affichés alternent noir sur fond blanc, blanc sur fond noir. Utiliser NORMAL pour revenir au mode d'affichage courant blanc sur fond noir.

"FLECHE A DROITE"

"FLECHE A GAUCHE"

Les touches marquées avec ces flèches à droite et à gauche sont utilisées pour éditer. La → déplace le curseur vers la droite, chaque caractère traversé est considéré comme ayant été tapé par l'utilisateur.

La ← déplace le curseur vers la gauche, chaque caractère traversé est effacé de la mémoire de l'ordinateur et de la ligne du programme que vous êtes en train de taper.

```
FOR W = 1 TO 20...NEXT W
FOR Q = 2 TO -3 STEP -2...NEXT Q
FOR Z = 5 TO 4 STEP 3...NEXT
```

Permet de créer une boucle où les instructions entre le FOR et le NEXT seront répétées un certain nombre de fois. Dans le 1er exemple, la variable compte combien de fois les instructions sont exécutées, les instructions dans la boucle seront exécutées pour W valant 1, 2, 3...20 puis la boucle se termine (W valant 21) et l'instruction après le NEXT W s'exécute. Le deuxième exemple illustre comment utiliser le pas (STEP) quand le comptage se fait autrement que de 1 en 1. Le contrôle de comptage se fait en fin de boucle, donc dans le 3ème exemple, les instructions dans la boucle seront exécutées une fois.

FRE (0)

Calcule la place mémoire, en octets, qui reste disponible pour l'utilisateur. Ce que vous placez entre les parenthèses n'a pas d'importance, du moment que le B.E.V.F. peut l'interpréter.

GET REP%

Demande un caractère venant du clavier, l'assigne dans REP% sans l'afficher sur l'écran et sans que la touche **RETURN** soit tapée.

GOSUB 250

Provoque le saut du programme au sous-programme débutant à la ligne indiquée (250 dans l'exemple). Quand un RETURN est rencontré, l'ordinateur revient exécuter l'instruction du programme principal qui suit immédiatement l'instruction d'appel du sous-programme GOSUB.

GOTO 250

Provoque un branchement inconditionnel à la ligne indiquée (250 dans l'exemple) du programme.

GR

Sélectionne le mode graphique couleur large résolution (40 X 40) laissant 4 lignes de texte en bas de l'écran. L'écran est nettoyé, le curseur se place en haut à gauche de la fenêtre de texte. Et la couleur (COLOR) s'initialise à 0 (noir).

HCOLOR = 4

Fixe la couleur d'affichage en graphisme couleur haute résolution. Voici la liste des couleurs et le numéro correspondant:

0	:	noir 1	4	:	noir 2
1	:	vert (dépend de la TV)	5	:	(dépend de la TV)
2	:	magenta (" " ")	6	:	(" " " ")
3	:	blanc 1	7	:	blanc 2

HGR

Uniquement disponible avec la carte ROM du B.E.V.F.

Sélectionne le mode haute résolution couleur (360 X 160) plus 4 lignes de texte en bas de l'écran. L'écran est nettoyé et la page 1 de la mémoire s'affiche. HCOLOR et la page de texte ne sont pas modifiés par HGR.

Le curseur n'est pas déplacé dans la fenêtre de texte.

HGR2

Affiche la page 2 de la haute résolution (360 X 192). Pas de texte au bas de l'écran. L'écran est nettoyé et la page 2 de la mémoire s'affiche. La page de texte n'est pas modifiée.

HIMEM: 16384

Fixe l'adresse de la plus grande case mémoire disponible pour un programme B.E.V.F., variables incluses. Utilisé pour protéger une zone mémoire pour des données, des dessins haute résolution ou des sous-programmes en langage machine.

HIMEM: n'est pas modifié par RUN, CLEAR, NEW, DEL, modifier des lignes, ajouter des lignes ou **RESET**.

HLIN 10, 20 AT 30

Permet, en large résolution, de dessiner une ligne horizontale dans la dernière couleur (COLOR) déclarée. L'origine (X = 0, Y = 0) est le coin supérieur gauche de l'écran.

Dans l'exemple, la ligne est dessinée de X = 10 à X = 20 pour Y = 30. On aurait aussi pu dire: la ligne se trace du point (10, 30) au point (20, 30).

HOME

Déplace le curseur sur le coin supérieur gauche de la fenêtre de texte et nettoie entièrement celle-ci.

```
HPlot 10, 20
HPlot 30, 40 TO 50, 60
HPlot TO 70, 80
```

Dessine des points et des lignes en graphisme haute résolution dans la dernière couleur (HCOLOR) déclarée.

L'origine est le point ($X = 0$, $Y = 0$), coin supérieur gauche de l'écran. Le premier exemple dessine un point aux coordonnées $X = 10$, $Y = 20$. Le second exemple trace une ligne reliant les points ($X = 30$, $Y = 40$) et ($X = 50$, $Y = 60$). Enfin le troisième exemple trace une ligne partant du dernier point dessiné et rejoignant le point ($X = 70$, $Y = 80$), et dans la couleur du dernier point dessiné sur l'écran, pas forcément dans la couleur (HCOLOR) la plus récente.

HTAB 23

Déplace le curseur sur la colonne spécifiée (entre 1 et 40). L'exemple positionne le curseur sur la colonne 23.

```
IF AGE < 18 THEN A = 0 : B = 1 : C = 2
IF REP% = "YES" THEN GOTO 100
IF N < MAX THEN 25
IF N < MAX GOTO 25
```

Si l'assertion suivant IF est vraie, alors l'(les) instruction(s) suivant THEN sont exécutées. Autrement les instructions suivant THEN sont ignorées et l'exécution continue à la ligne suivante du programme. Les expressions alphanumériques sont comparées selon leur rang alphabétique. La syntaxe des instructions de branchement est identique pour les exemples 2, 3 et 4, malgré la présentation différente.

```
INPUT A%
INPUT "DONNEZ LA DATE"; C%
```

Dans le premier exemple, INPUT affiche un point d'interrogation et attend un nombre, qui sera assigné dans la variable entière A%. Dans le second exemple INPUT affiche la chaîne "DONNEZ LA DATE" et attend sans afficher de point d'interrogation que vous donniez une chaîne alphanumérique qui sera assignée dans C%. Des entrées multiples sur un INPUT doivent être séparées par des virgules ou des RETURN.

INT (NUM)

Calcule la partie entière de l'argument (NUM dans l'exemple). Si NUM vaut 2.34 alors l'ordinateur renverra 2. (NUM = 5.34 l'ordinateur renverra -6)

INVERSE

Inverse l'affichage TV, les caractères venant de l'ordinateur sont affichés noirs sur fond blanc. Utilisez NORMAL pour revenir à l'affichage courant, blanc sur fond noir.

IN # 4

Choisit le connecteur E/S (entre 1 et 7) du périphérique d'entrée qui sera mis en fonction IN # 0 rend le contrôle d'entrée au clavier.

LEFT% ("AVION", 3)

Retourne le nombre spécifié de caractères de la chaîne alphanumérique. Dans l'exemple, ce sera AVI (les 3 caractères les plus à gauche).

LEN ("BONJOUR")

Retourne le nombre de caractères d'une chaîne alphanumérique, entre 0 et 255. Dans l'exemple, ce nombre sera 7.

LET A\$ = "GRAND"

A = 23.567

Assigne au nom de la variable à gauche de = la valeur ou la chaîne à droite de =
LET est optionnel.

LIST

LIST 200 - 3000

LIST 200, 3000

Le premier exemple provoque l'affichage de tout le programme sur l'écran, le second et le troisième provoquent l'affichage des lignes du programme dont le numéro est compris entre 200 et 3000.

Pour lister du début du programme à la ligne 200 tapez LIST -200, pour lister de la ligne 200 à la fin du programme, utilisez LIST 200-

CTRL arrête un listing.

C

LOAD

Lit un programme B.E.V.F. sur la cassette et le charge dans la mémoire de l'ordinateur. Aucun message ne s'affiche, l'utilisateur doit rembobiner la cassette et appuyer sur la touche de lecture. Un "bip" sonore indique que le programme est retrouvé. À la fin du chargement un second "bip" se produit et le signe de reconnaissance (!) réapparaît. Seul **RESET** peut interrompre un LOAD.

LOG (2)

Calcule le logarithme népérien de l'argument.

L'exemple retourne -693147181

LOMEM: 2060

Fixe l'adresse de la plus petite case mémoire disponible pour un programme B.E.V.F. Cela permet de protéger les variables des dessins haute résolution dans les systèmes dont la configuration mémoire est importante.

MID\$ ("AVION", 4)

MID\$ ("AVION", 3, 2)

Extrait une sous-chaîne d'une chaîne alphanumérique. L'exemple 1 renvoie la sous-chaîne comprise entre le 4ème et le dernier caractère: ON. L'exemple 2 renvoie une sous-chaîne de 2 caractères à partir du 3ème, c'est-à-dire IO.

NEW

Détruit le programme en mémoire et les variables.

NEXT

Voir FOR...TO...STEP

NORMAL

Fixe la mode d'affichage normal: caractères blancs sur fond noir.

NOTRACE

Arrête le mode dépistage, voir TRACE.

ON ID GOSUB 100, 200, 23, 400, 500

Exécute un sous-programme à la ligne indiquée par la valeur de l'expression arithmétique suivant ON. Dans l'exemple si ID = 1 alors GOSUB 100 est exécuté, si ID = 2 alors GOSUB 200 est exécuté. Et ainsi de suite. Si la valeur de l'expression est 0 ou est supérieure au nombre de lignes, l'exécution continue à l'instruction suivante.

ON ID GOTO 100, 200, 23, 400, 500

Identique à ON ID GOSUB (ci-dessus) sauf que c'est un branchement incondicional GOTO qui s'exécute.

ONERR GOTO 500

Utilisé pour éviter un message d'erreur et l'arrêt d'exécution du programme. A l'exécution, ONERR GOTO positionne un indicateur de sortie qu'un branchement incondicional (GOTO 500 dans l'exemple) exécute en cas de détection d'erreur.

PDL (3)

Retourne la valeur (entre 0 et 255) du levier de commandes n° 3 dans l'exemple. L'argument doit être compris entre 0 et 3.

PEEK (37)

Retourne le contenu, en décimal, de l'octet de l'adresse décimale spécifiée (37 dans l'exemple).

PLOT 10, 20

En graphisme large résolution, dessine un petit rectangle aux coordonnées spécifiées (X = 10, Y = 20 dans l'exemple).

La couleur du rectangle est déterminée par le dernier COLOR.

POKE -16302, 0

Place l'équivalent binaire de la valeur décimale du deuxième argument (0) dans la case mémoire dont l'adresse est spécifiée par le premier argument (-16302).

POP

Provoque un saut d'un niveau de retour de sous-programme. La pile de retour (RETURN) est descendue d'un cran.

POS (0)

Retourne la position horizontale du curseur. C'est un chiffre de 0 (marge gauche) à 39 (marge droite). Ce que vous avez mis entre parenthèses n'a pas d'importance, tant que le B.E.V.F. peut l'interpréter.

PRINT

PRINT A\$; "X = " ; X

Le premier exemple provoque un saut de ligne. Les articles d'une liste dans un PRINT doivent être séparés de virgules (ou être tous dans un champ de tabulation différent). Ils devront être séparés par un point-virgule pour qu'ils s'affichent collés les uns aux autres.

Si A\$ = "CORE" et X = 3, l'exemple 2 donnera sur l'écran:

COREX = 3

PR # 2

Donne la sortie au connecteur E/S spécifié (entre 1 et 7). PR # 0 redonne la sortie sur l'écran télé.

READ A, B%, C%

A l'exécution assigne aux variables de l'instruction READ, les valeurs successives des éléments dans les DATA du programme.

Dans l'exemple: les deux premiers éléments des DATA doivent être des nombres et le dernier doit être une chaîne alphanumérique.

RECALL MX

Charge un tableau numérique qui a été stocké sur cassette. Le nom du tableau n'a pas d'importance. Quand il est appelé, MX doit avoir été préalablement dimensionné. On indique par l'index dans STORE ou RECALL. Dans l'exemple, les éléments MX (0), MX (1)... seront chargés. Aucun message n'est affiché. L'utilisateur doit actionner son magnétophone et le "bip" sonore signale le début et la fin du tableau. Seul **RESET** peut interrompre RECALL.

REM C'EST UNE REMARQUE

Permet l'insertion de remarques dans le programme.

REPT

Si vous appuyez sur **REPT**, et pressez en même temps sur un autre caractère, ce dernier sera répété.

RESTORE

Réinitialise au premier élément le pointeur de liste de données (DATA). L'exécution de READ enchaînera la lecture du 1er élément de la table des DATA.

RESUME

A la fin du programme de manipulation des erreurs (voir ONERR GOTO), provoque la continuation du programme à l'instruction qui avait provoqué l'erreur.

RETURN

Indique la fin d'un sous-programme et provoque le retour à l'instruction suivant le dernier GOSUB appelé.

RIGHT% ("AVION", 3)

Retourne le nombre de caractères spécifiés les plus à droite de la chaîne alphanumérique. Dans l'exemple on aura: 10N.

RND (5)

Calcule un nombre aléatoire réel plus grand ou égal à 0 mais plus petit que 1. RND (0) retourne le dernier nombre aléatoire tiré. Chaque argument négatif retourne un nombre aléatoire qui sera le même à chaque fois que RND est utilisé avec cet argument, et les RND suivants avec des arguments positifs auront une séquence périodique. Chaque fois que RND est utilisé avec un argument positif, un nouveau nombre aléatoire est tiré, jusqu'à ce qu'une séquence périodique s'initialise à cause d'un argument négatif.

ROT = 16

Fixe la rotation angulaire d'une figure haute résolution, dessinée ou à dessiner par DRAW ou XDRAW. ROT = 0 provoque l'affichage de la figure telle qu'elle a été conçue. ROT = 16 tourne la figure de 90° dans le sens des aiguilles d'une montre, etc.

La période de rotation est de 64.

RUN 500

Efface les variables, initialise les pointeurs et les piles et commence l'exécution du programme au numéro de ligne indiqué. S'il n'y a pas de numéro, le programme est exécuté

à partir de sa première ligne.

SAVE

Conserve un programme sur cassette. Aucun message n'est indiqué, l'utilisateur doit mettre son magnétophone sur le mode enregistrement avant de conserver le programme. SAVE ne vérifie pas si le magnétophone est bien connecté.

Un "bip" sonore signale le début et la fin de l'enregistrement.

SCALE = 50

Fixe le facteur d'échelle d'une figure haute résolution qui sera dessinée par XDRAW ou DRAW. SCALE = 1 reproduit la taille de la figure originelle.

SCALE = 255 agrandit 255 fois chaque vecteur point.

SCALE = 0 est la taille maximale de la figure.

SCRN (10, 20)

En graphisme large résolution, retourne la couleur du point spécifié. Pour cet exemple, c'est le point X = 10, Y = 20.

SGN (NUM)

Retourne 1 si l'argument est positif, -1 s'il est négatif et 0 s'il est nul.

SHLOAD

Charge une table de construction de figures dans la cassette. La table est chargée juste en dessous de HIMEM: puis HIMEM: est fixé juste en dessous de la table.

SIN (2)

Calcule le sinus de l'argument, qui doit être en radians. L'exemple retourne: .909297427

SPC (8)

Doit être utilisé dans une instruction PRINT. Introduit le nombre spécifié d'espaces entre les articles à droite et à gauche de SPC. 8 espaces sont laissés dans l'exemple.

SPEED = 50

Fixe la vitesse d'affichage des caractères sur l'écran ou la vitesse de communication des caractères aux autres périphériques. 0 est la vitesse la plus lente, 255 la plus rapide.

SQR (2)

Calcule la racine carrée de l'argument. Dans l'exemple 1.41421356 est renvoyé. Cette fonction s'exécute plus rapidement que $\Lambda.5$

STOP

Arrête l'exécution d'un programme et indique le numéro de ligne de l'arrêt. Le contrôle de l'ordinateur est rendu à l'utilisateur.

STORE MX

Enregistre sur cassette un tableau numérique. Aucun message n'est affiché, l'utilisateur doit mettre son magnétophone en mode enregistrement. Un "bip" sonore signale le début et la fin de l'enregistrement. Les index du tableau ne doivent pas apparaître dans STORE.

Dans cet exemple, les éléments MX (0), MX (1)... sont conservés sur cassette.

Voir RECALL.

STR\$ (12.45)

Retourne la chaîne représentant la valeur de l'argument "12.45" dans l'exemple.

TAB (23)

Doit être utilisé dans une instruction PRINT, l'argument doit être compris entre 0 et 255, si l'argument est supérieur à la valeur de la position du curseur, le curseur se déplace à la position indiquée, en comptant à partir de la marge gauche de la ligne du curseur. Si l'argument est inférieur à la valeur de la position du curseur, le curseur n'est pas déplacé. TAB (0) est le curseur en position 256.

TAN (2)

Calcule la tangente de l'argument, qui doit être en radians. Dans l'exemple, -2.18503987 est retourné.

TEXT

Sélectionne le mode texte de l'écran, 40 caractères par ligne et 24 lignes. **RESET** réinitialise la fenêtre de texte à tout l'écran.

TRACE

Sélectionne le mode de dépiage.

Affiche sur l'écran le numéro de ligne de chaque instruction exécutée. TRACE est toujours valable après un RUN, CLEAR, NEW, DEL ou **RESET**.

L'instruction NOTRACE coupe TRACE.

USR (3)

Cette fonction communique l'argument à un sous-programme en langage machine. L'argument est calculé et mis dans l'accumulateur à virgule flottante (de l'adresse 9D à A3), puis un JSR est exécuté à l'adresse 0A. Les adresses 0A, 0B, 0C doivent contenir un JMP à la première adresse du programme en langage machine. La valeur de retour pour cette fonction est mise dans l'accumulateur à virgule flottante. Pour revenir en B.E.V.F. faites un RTS.

VAL ("-3.7E4A5PLE")

Essaie d'interpréter une chaîne jusqu'au premier caractère non-numérique comme valeur réelle ou entière et redonne la valeur de ce nombre. S'il n'y a pas de chiffre avant le premier caractère non-numérique, 0 est affiché. Dans l'exemple ci-dessus, -37000 sera retourné.

VLIN 10, 20 AT 30

Instruction, utilisée dans le mode graphique large résolution, qui dessine une ligne verticale dans la couleur fixée par la dernière instruction COLOR. La ligne se dessine dans la colonne, indiquée par le 3ème argument. Dans cet exemple, la ligne va de Y = 10 à Y = 20 dans la colonne X = 30.

VTAB (15)

Déplace le curseur sur la ligne de l'écran spécifiée par l'argument. Le numéro de la ligne du bout de l'écran est 1, et du bas de l'écran 24.

VTAB déplace le curseur vers le haut ou vers le bas mais ni vers la gauche, ni vers la droite.

WAIT 16000, 255

WAIT 16000, 255, 0

Donne la possibilité d'insérer dans un programme une pause conditionnelle. Le premier argument est l'adresse décimale d'une case mémoire devant être testée pour voir si certains bits sont à 1 (ou haut) et si d'autres bits sont à 0 (ou bas). La valeur donnée à chaque bit de l'équivalent binaire du second argument permet de sélectionner les bits de

la case mémoire qui vous intéressent. Un bit à 1 du 2ème argument indique que le bit de même ordre de la case mémoire vous intéresse. Un bit à 0 du 2ème argument indique que le bit de même ordre de la case mémoire ne vous intéresse pas. Chaque bit de l'équivalent binaire du 3ème argument indique dans quel état (0 ou 1) vous attendez (WAIT) le bit de même ordre de la case mémoire testée: 1 indique que le bit doit être à 0 (bas) 0 indique que le bit doit être à 1 (haut). Si le troisième argument n'est pas spécifié, il est considéré à zéro par défaut. Si un des bits correspond au bit de l'argument 2 et perd la valeur indiquée par l'argument 3 la pause (WAIT) s'arrête.

XDRAW 3 AT 180, 120

Utilisée dans le mode graphisme haute résolution, cette instruction dessine à X = 180, Y = 120, la figure numéro 3 d'une table de figures qui a été préalablement enregistrée. Chaque point affiché est visible dans la couleur du complément de la couleur déjà existante à ce point.

Donc, utiliser cette instruction pour effacer les figures. Une première instruction XDRAW dessine une figure, une deuxième fois XDRAW efface cette figure sans changer la couleur du fond.



Notes

ANNEXE P : INDEX ALPHABÉTIQUE GÉNÉRAL

174

A

ABS page 105
Accélérer (exécution des programmes) page 122
Adresse pages 46-47-49 à 51-122
Affichage TV page 58
Aléatoire page 105
AND
Arrêter un programme annexe G (p. 126)
Arrondis page 145
pages 14-15-27-28-38 à 40
ASC pages 63-162
ASCII (codes des caractères) annexe K (p. 144 à 146)
Assertion pages 19-20
Assignation (instruction) pages 17-18
Astérisque pages 12-107
AT pages 16-33-87-163-171-172
ATN page 105

B

BASIC page 10
BASIC évolué (B.E.) page 10
BASIC (chargement) page 107
BASIC Entier (par rapport au B.E.V.F.) annexe M (p. 153 à 155)
B.E.V.F. annexe A (p. 107)
chargement annexe A (p. 107)
conversion en annexe H (p. 129-130)
par rapport au BASIC Entier annexe M (p. 153 à 155)
sur carte ROM pages 107 à 109
sur cassette pages 109 à 111
Boucles: voir FOR...NEXT pages 21-23-28 à 29-78 à 80-164
Boucles d'attente pages 35-47 à 49-100-101
Branchement: boucles
GOSUB pages 25-80-165
GOTO pages 77-165

C

CALL page 49-annexe J

Caractères ASCII codes	annexe K (p. 144 à 146)
chaînes	pages 28 à 32
Caractères alphanumériques	pages 28 à 32
Caractère de reconnaissance	chapitre 2 (p. 37)
Carte ROM (B.E.V.F.)	pages 107 à 109
Cassette: tableaux	page 24
tables de construction des figures	chapitre 9 (p. 93 à 103)
chargement du B.E.V.F.	page 107
place mémoire	page 119
Chaînes	pages 28 à 32-141
ASC	pages 63-162
CHR\$	pages 62-162
concaténation	page 129
conversion en B.E.V..F	pages 26-70-71-163
DATA	pages 19-20-166
IF...THEN	pages 68-151-166
INPUT	pages 29-63-129-166
LEFT\$	pages 28-62-167
LEN	pages 17-21-74-167
LET	pages 12-17-45 à 48
mémoire	pages 29-64-166
MID\$	page 28
chaîne nulle	pages 64 à 67
RECALL	page 169
RIGHT\$	pages 29-64-169
STORE	pages 64 à 67-170
STR\$	pages 28 à 31-62-170
sous-chaîne	pages 63-64
VAL	pages 29 à 31-62-171
Chaîne nulle	page 28
ASC	pages 63-162
DATA	pages 26-70-71-163
IF...THEN	pages 19-20-166
INPUT	pages 68-151-166
MID\$	pages 29-64-166
Chargement du BASIC	page 107
Changement d'une ligne de programme	pages 59-11 à 115
Chiffres	pages 14-27 à 31
Chiffres (significatifs)	page 14
CHR\$	pages 62-162
Clavier	annexe J (p. 134)
CLEAR	pages 18-57-162
Code hexadécimal	annexe K (p. 144)
Code des caractères de contrôle	annexe J (p. 134)
Colonne: voir champ de tabulation	pages 55-56-72-73
Commandes	pages 12-13
Concaténation	page 129
conversion en B.E.V.F.	annexe H (p. 129-130)
PRINT	pages 12-16-72-73-168
SPC	pages 56-57-170

CONT		pages 45-69-162
Control B		pages 107 à 109
Control C	DATA	pages 26-70-71-163
16-20-41	GET	pages 69-70-165
45-46-109	INPUT	pages 68-166
à 111-163	LIST	pages 12-13-53-54-167
Control H		page 69
Control M		pages 68-70
Control X		pages 60-68-70-163
Conversion en B.E.V.F.		annexe H (p. 129-130)
COS		pages 105-163
Cosinus (fonction): voir COS		pages 105-163
COLOR		pages 15-20-32-33-86-162
CTRL (control)		pages 141-156
Curseurs (positions)		pages 55 à 57-59-60-109 à 111-134-136
D		
DATA		pages 26-70-71-163
DEF		pages 27-74-75-163
Définitions syntaxiques		pages 37 à 42-annexe N (p. 156 à 160)
DEL		pages 54-163
Dépistage (mode)		pages 45-46
Déplacement du curseur		pages 55 à 57-59-60-109 à 111-134-136
Dessiner		chapitre 9 (p. 93 à 103)
Deux points (:)		pages 37 à 42-68 à 71
	DATA	pages 26-70-71-163
	GET	pages 32-69-70-165
	INPUT	pages 16-19-68-151-166
Différence entre le B.E.V.F. et le BASIC Entier		annexe M (p. 153 à 155)
DIM		pages 24-61-163
Dimension: voir DIM		
Divisions (/)		pages 12-27-37 à 42
DRAW		pages 93 à 101-163
Droite (flèche à)		pages 59-111 à 113-164
E		
Editer		chapitre 4 (p. 53)-annexe B (p. 111)
Effacer	programme	pages 13-44-54
	écran	pages 18-57
Egal (signe)		pages 18-21-22
Elément	tableaux	pages 24-37 à 42-61-64 à 67
	DATA	pages 26-70-71-163
END		pages 25-45-119-120-163
Entrée-sortie		pages 44-64 à 75

levier de commandes et haut-parleur
Entier calcul
12-14 fonction INT
arrondis
variables
Erreur ONERR GOTO
ESC **ESC** **A**, **B**, **C**, **D**,
ESC **E**, **F**,
Exécution
Exécution (interruption)
Exécution (en mode programme)
EXP
expr
expra
exprc
Exposant
Exponentielle (fonction) voir: EXP

F

Fenêtre de texte
Figures
Fixe (notation en virgule fixe)
Fixer la marge de la fenêtre de texte
FLASH
Flèche (touche de)
Flottante (notation en virgule flottante)
FN
Fonction
Format
Format des nombres
FOR...NEXT
FRE

G

Gagner de la place mémoire
Gauche (flèche à)
GET
GOSUB...RETURN
GOTO
GR
Graphisme total
Graphisme (haute résolution)
Graphisme (large résolution)
Guillemets
DATA
INPUT
chaînes

pages 91-139
page 42
pages 27-105-166
pages 14-15-27-28-38 à 40
pages 17-18-27-28-141
pages 81-82-142-151-168
pages 59-111 à 115
pages 12-43
page 45
pages 12-43
pages 27-106-164
chapitre 2 (p. 37)-annexe N (p. 156)
pages 14-27-37 à 42

pages 55-56-72-73-85-134-135
chapitre 9 (p. 93 à 103)
page 14
page 134
pages 58-164
pages 59-111 à 113-164
page 14-annexe E-pages 122-151
pages 74-75-163 (voir DEF FN)
pages 27-74-75-105-106-163
pages 14 à 16-27
page 14
pages 21 à 23-28-29-78 à 80-164
pages 57-58-164

pages 119-120
pages 59-111 à 113-164
pages 32-69-70-165
pages 25-80-120-165
pages 16-77-165
pages 15-20-32-85-137-165
chapitre 8 (p. 85 à 91)-137
pages 21-22-33 à 35-93 à 103
pages 15-20-32-33-85-91-131-137
pages 26-70-71-163
pages 68-151-166
pages 28 à 32

H

Haut-parleur	pages 91-139
Haute résolution (graphisme en)	pages 21-22-33 à 35-93 à 103
stockage mémoire	page 120
page zéro	annexe L (p. 150 à 152)
HCOLOR	pages 33-90-137-165
Hexadécimal (code)	page 144
HGR	pages 32-33-85-88-165
HGR2	pages 32-89-165
HIMEM:	pages 47-49-103-126-127-132-165
HLIN	pages 15-32-87-165
HOME	pages 20-57-166
HPlot	pages 33-36-90-137 à 142-166
HTAB	pages 33 à 36-55-166

I

IF...GOTO	pages 77-78-166
IF...THEN	pages 19-20-77-166
Immédiat (mode d'exécution)	pages 12-43
Incrément dans les boucles	pages 22-78-79
Index	page 187
Insertion	pages 47-48
pauses	pages 12-14-113
textes	pages 113 à 115
Insérer des lignes	pages 20-129-130
Instructions multiples sur une ligne	pages 68-151-166
INPUT	pages 27-105-166
INT	page 45
Interruption d'exécution	pages 58-166
INVERSE	pages 27-105-106
INVERSE (fonction trigonométrique)	pages 73-166
IN#	pages 27-105-106-119-120
Internes (programmes)	pages 21 à 23
Itération	

J

Jeu (levier de jeu)	page 91-annexe J(p. 137)
---------------------	--------------------------

L

Large résolution (graphiques)	pages 15-20-32-33-85-91-131-137
LEFT\$	pages 29-63-129-166
LEN	pages 28-62-167
LET	pages 17-21-74-167
Leviers de commandes et de jeu	chapitre 8 (p. 85)

Ligne	pages 12-43-119-151
Ligne en mode graphique	page 87-chapitre 9 (p. 93)
Ligne (saut de)	pages 72-73-annexe J (p. 137...)
Ligne (numéro)	pages 12-37-54-156...
taille en octet	page 119
DATA	pages 26-70-71-163
GOTO	pages 77-165
LIST	pages 53-54-167
ON...GOTO	pages 81-82-168
page zéro	annexe L (p. 150 à 152)
LIST	pages 12-13-53-54-167
Littéral	pages 28-37 à 42-156...
DATA	pages 26-70-71-163
INPUT	pages 68-151-166
LET	pages 17-21-74-167
LOAD	pages 44-167
Logarithme: fonctions	(voir LOG)
LOG	pages 27-106-167
LOMEM:	pages 50-126-127-132-167

M

Mantisses	page 12
Marge (fixe la marge de la fenêtre de texte)	page 134
MAT (conversion en B.E.V.F.)	annexe H (p. 129)
Matrice: voir tableaux	
Métanom	pages 37-156...
Métasymbole	pages 37-156...
MID\$	pages 29-64-166
conversion en B.E.V.F.	annexe h (p. 129)
MODE de dépistage	page 46
MODE programme (exécution en)	pages 12-43
MODE immédiat (exécution en)	pages 12-43
Mémoire	
emplacement des messages d'erreurs	pages 116 à 118
HGR	pages 32-33-85-88-165
HGR2	pages 32-89-165
cartes	annexe I (p. 131-132)
comment gagner de la place mémoire	pages 119-120
page zéro	annexe L (p. 150 à 152)
Mot du langage (valeurs décimales)	page 124
Mots réservés	pages 126-127
LIST	pages 53-54-167
stockage mémoire	page 120

N

NEW
 NEXT
 Nom, nom%, nom \$
 Nombre aléatoire (fonction) voir: RND

NORMAL
 NOT
 Notations scientifiques

NOTRACE
 Numligne

pages 12-44-167
 pages 21 à 23-28-29-78 à 80-167
 pages 37 à 42-156 à 160

pages 58-167
 pages 37 à 42
 page 12

pages 46-167
 pages 12-37 à 42-156 à 160

O

ON...GOSUB
 ON...GOTO
 ONERR GOTO
 op
 opa
 opal
 oploc
 opc
 Opérateurs arithmétiques

OR

pages 81-82-168

pages 37 à 42-156 à 160

pages 37 à 42

P

Page zéro
 Pause
 PDL
 PEEK
 Périphériques
 Place de la virgule

PLOT
 POKE
 graphisme total
 Point d'interrogation

INPUT
 PRINT
 Pointeurs

POP

Port E/S Ø
 POS

Position du curseur
 Précision des nombres
 Priorité des opérateurs
 Programme
 pointeurs de page zéro

annexe L (p. 150 à 152)
 pages 47-48

pages 91-168
 page 46-annexe J-168
 pages 73-91-131-annexe J (p. 137)
 page 14-annexe E (p. 122)-151

pages 15-20-32-86-168
 page 47-annexe J-168
 chapitre 8 (p. 85 à 91)-137

pages 68-151-166
 pages 12-16-72-73-168
 pages 57-71-80-131-132-141-150 à 152

pages 81-168

(voir pages 44-64 à 75)
 pages 56-168
 pages 55 à 57-59-60
 pages 14-15

page 42
 page 12
 annexe L (p. 150 à 152)

PRINT	pages 12-16-72-73-168
chaînes	pages 28 à 32
TAB	pages 56-71
SPC	pages 56-57-170
PR#	pages 73-168
R	
Racine carrée (fonction) voir: SQR	
READ	pages 26-71-151-169
RECALL	pages 64 à 67-169
Reconnaissance (des caractères)	chapitre 2 (p. 37)
Réels	pages 12-37 à 42-141-142
calculs	page 27
DATA	pages 26-70-71-163
noms des variables	page 17
Relation entre les expressions	pages 19-42
REM	pages 17 à 20-55-119-120-169
Répétitions (REPT)	pages 59-111 à 113-169
Réservés (mots)	pages 126-127
LIST	pages 12-13-53-54-167
stockage mémoire	page 120
RESET	pages 37 à 42-45
HIMEM:	pages 47-49-103-126-127-132-165
LOMEM:	pages 50-126-127-132-167
RECALL	pages 64 à 67-169
RESUME	pages 83-169
STORE	pages 64-65-170
RESTORE	pages 26-72-169
RESUME	pages 83-169
RETURN (touche RETURN)	pages 12-16-37 à 42
GET	pages 32-69-70-165
INPUT	pages 68-151-166
PRINT	pages 12-16-72-73-168
RETURN	pages 25-80-169
RIGHTØ	pages 29-64-169
RND	pages 27-33 à 36-105-151
ROM - B.E.V.F.	pages 107 à 109
ROT	pages 93 à 103-169
RUN	pages 12-18-44-169
Résumé des commandes du B.E.V.F.	annexe O (p. 162 à 172)
S	
Saut de ligne	pages 72-73-annexe J (p. 137)
SAVE	pages 44-170
SCALE	pages 93 à 103-170
SCRN	pages 86-87-170

Scientifique (notation)	page 12
Séparateur	pages 37 à 42-68-71
SGN	pages 105-170
SHLOAD	pages 100-103-170
Significatifs (chiffres)	page 14
Signe: voir SGN	
SIN	pages 27-105-170
Sortie (mode TV)	page 58
Sous-programme	pages 25-28 à 30-80
Sous-programme (langage machine)	pages 49 à 51-93 à 101
Sous-chaîne	pages 63-64
SPC	pages 56-57-170
SPEED	pages 58-170
SQR	pages 21-22-27-105-170
STEP	(voir FOR...NEXT)
STOP	pages 25-45-170
Stockage mémoire	page 120
STORE	pages 64 à 67-170
STR\$	pages 29-30-62-170
Supprimer	(voir Effacer)
Symboles spéciaux	page 37
Syntaxiques (définitions alphabétiques)	annexe N (p. 156 à 160)
T	
TAB	pages 56-171
champ de tabulation	pages 55-56-72-73
HTAB	pages 33 à 36-55-166
TAB	pages 56-171
VTAB	pages 55-171
Tableaux	pages 24-chapitre 5 (p. 61)-134
place mémoire utilisée	page 121
carte de mémoire	page 131
STORE, RECALL	
comment gagner de la place mémoire	pages 119-120
page zéro	annexe L (p. 150 à 152)
TAN	pages 27-105-171
Tangente (fonction): voir TAN	pages 27-105-171
Télévision (affichage)	page 58
TEXT	pages 16-20-85-171
Texte	pages 16-32
graphique	pages 20-137 à 142-171
stockage mémoire	page 120
fenêtre de texte	pages 55-56-72-73-85-134-135
THEN: voir IF...THEN	
TO: voir PLOT et GOTO	
TRACE	pages 46-81-82-171

Tri	pages 24-30 à 32
Trigonométriques (fonctions)	pages 27-105-106
U	
USR	pages 50-51
V	
VAL	pages 29 à 31-62-171
Valeurs décimales (des mots de langage)	pages 124-125
Valeurs absolues (fonction) voir: ABS	
var	pages 37 à 42-156 à 160
vara	
varc	
Variables	
tableau	pages 17-18-27-28-141
boucles FOR...NEXT	pages 134-141
INPUT	(voir FOR...NEXT)
entier	pages 68-151-166
LET	pages 12-14-27-28
noms	pages 17-21-74-167
vitesse de programme	pages 17-18-37 à 42-156 à 160
READ, DATA	annexe E (p. 122)
réel	pages 26-70-71-151-163-169
comment gagner de la place mémoire	pages 27-28-37 à 42-141-156 à 160
chaîne	pages 119-120
page zéro	pages 28 à 32-141
Variables prête-nom	annexe L (p. 150 à 152)
Vecteur dessin	page 75
Virgule flottante (notation en)	pages 93 à 103
Virgule fixe (notation en)	page 14-annexe E (p. 122)-151
Virgule	page 14
DATA	pages 26-70-71-163
GET	pages 32-69-70-165
INPUT	pages 68-151-166
PRINT	pages 12-16-72-73-168
VLIN	pages 15-33-87-171
VTAB	pages 33 à 36-55-171
W	
WAIT	pages 47-48-171
X	
XDRAW	pages 102-172
XPLOT	page 127
Z	
Zéro (page)	annexe L (p. 150 à 152)

MESSAGES D'ERREURS	pages
?BAD SUBSCRIPT	117
DIM	61
?CAN'T CONTINUE	116
CONT	45
?DIVISION BY ZERO	116
?EXTRA IGNORED	
GET	69
INPUT	68
?FORMULA TOO COMPLEX	117
IF	19
?ILLEGAL DIRECT	116
INPUT	68
?ILLEGAL QUANTITY	116
ASC	63
CALL	49
CHR\$	62
DRAW	93
HIMEM:	47
HPLOT	90
HTAB	55
IN#	73
LEFT\$	63
MID\$	64
ON...GOSUB	81
ON...GOTO	81
PDL	91
PLOT	86
POKE	47
RIGHT\$	64
SPC	56
SPEED	58
STORE, RECALL	64
VLIN	87
VTAB	55
WAIT	47
?NEXT WITHOUT FOR	116
FOR	78
NEXT	78
?OUT OF DATA	117
READ	71
RECALL	64
STORE	64
?OUT OF MEMORY	117
DIM	61
GOSUB	80
HIMEM:	47
LOMEM:	50

?OVERFLOW ERROR	117
Réels	37
STR\$	62
VAL	62
?REDIM'D ARRAY	117
DIM	61
?REENTER	
INPUT	68
?RETURN WITHOUT GOSUB	117
RETURN	80
?STRING TOO LONG ERROR	117
LEN	62
PRINT	72
VAL	62
?SYNTAX ERROR	117
ASC	63
CONT	45
DATA	70
DEL	54
FOR...NEXT	78
GET	69
HGR	32
HGR2	32
IF...THEN	77
INPUT	68
LIST	53
RECALL	64
RESUME	83
RUN	44
SHLOAD	100
STORE	64
TEXT	85
?TYPE MISMATCH	118
LEFT\$	63
LET	74
MID\$	64
RIGHT\$	64
?UNDEF'D FUNCTION	118
DEF	74
?UNDEF'D STATEMENT	118
GOSUB	80
GOTO	77
RUN	44

INDEX DES SIGNES

"		pages 19-37 à 42-68-69-70-72-73
§		pages 27-37 à 42-62-63
‰		pages 27-37-39
*		pages 12-37-42-107
+		pages 12-37 à 42-68-70
-		pages 12-37 à 42-68-70
,		pages 12-16-37 à 42-68 à 73
/		pages 12-37 à 42-130
:		pages 37 à 42-68-70
;		pages 16-37 à 42-68 à 73
?		pages 16-37-68-72
\		page 37
]]		pages 37 à 42-107
^		pages 37 à 42
		page 37
~		pages 37-39
()		pages 24-37-39-119-120
[]		page 37
{ }		page 37
=	pour l'assignation	pages 17-21
>	comme caractère de reconnaissance	page 107
=, >, <		pages 19-37-39-42
&		pages 126-128

ANNEXE Q :

INDEX DES COMMANDES

	pages		pages		pages
ABS	105	INVERSE	58	SPC	56
ASC	63	IN#	73	SPEED	58
ATN	105	LEFT\$	29	SQR	21
CALL	49	LEN	28	STEP	78
CHR\$	62	LET	17	STOP	25
CLEAR	57	LIST	53	STORE	64
COLOR	86	LOAD	44	STR\$	29
CONT	69	LOG	106	TAB	56
COS	105	LOMEM:	50	TAN	105
CTRL	C	MID\$	29	TEXT	85
CTRL	X	NEW	12	Touche flèche	111
DATA	70	NEXT	78	TRACE	46
DEF FN	74	NORMAL	58	USR	50
DEL	54	NOTRACE	46	VAL	29
DIM	61	ON...GOSUB	80	VLIN	87
DRAW	93	ON...GOTO	80	VTAB	55
END	45	ONERR GOTO	80	WAIT	47
ESC	A	PDL	91	XDRAW	102
ESC	B	PEEK	46		
ESC	C	PLOT	15		
ESC	D	POKE	47		
EXP	106	POP	81		
FOR... TO...STEP	78	POS	56		
FLASH	58	PRINT	72		
Flèche à droite	111	PR#	73		
Flèche à gauche	111	READ	26		
FRE	57	RECALL	64		
GET	69	REM	17		
GOSUB	80	REPT	111		
GOTO	77	RESET	7		
GR	85	RESTORE	26		
HCOLOR	90	RESUME	83		
HGR	32	RETURN	12		
HGR2	32	RIGHT\$	29		
HIMEM:	47	ROT	102		
HLIN	32	RND	27		
HOME	57	RUN	12		
HPLOT	33	SAVE	44		
HTAB	33	SCALE	102		
IF...GOTO	77	SCRN	86		
IF...THEN	77	SGN	105		
INPUT	68	SHLOAD	100		
INT	27	SIN	105		

imprimé en France-IF Paris
RC PARIS B 300960614



FRANCE

ITT OCEANIC
97 Avenue de Verdun
93230 - Romainville

BENELUX

BELL TELEPHONELLAN
2440 - GEEL
BELGIQUE