



MANUEL SBASIC
SOUS NEW DOS 80



MANUEL SBASIC

SOUS NEW DOS 80



Copyright version originale © 1984

Toute reproduction, même partielle, par quelque procédé que ce soit, est interdite sans autorisation écrite préalable. Une copie ou reproduction par xérographie, photographie, film, bande magnétique ou autre constitue une contrefaçon passible des peines prévues par la loi sur la protection des droits d'auteur.

Printed in France. World rights reserved.

SOMMAIRE

- 1 - INTRODUCTION AU SBASIC
- 2 - CHARGEMENT DEPUIS LE DOS
- 3 - LES SYMBOLES, CONSTANTES ET VARIABLES
- 4 - LES COMMANDES GENERALES
- 5 - LES INSTRUCTIONS GENERALES
- 6 - LES INSTRUCTIONS ECRAN/IMPRIMANTE/SON
- 7 - LES FONCTIONS MATHEMATIQUES
- 8 - LES FONCTIONS SUR CHAINE DE CARACTERES
- 9 - LES FICHIERS EN SBASIC
- 10 - LE TRAITEMENT DES ERREURS
- 11 - L'EDITEUR PLEINE PAGE

APPENDICES

- A - GLOSSAIRE TECHNIQUE
- B - INDEX ALPHANUMERIQUE DES COMMANDES SBASIC
- C - MESSAGES D'ERREURS COMMENTES
- D - LE CODE ASCII
- E - LE GENERATEUR DE CARACTERES GUEPARD
- F - LES FICHIERS (RECAPITULATIF)

PREAMBULE

Ce manuel n'est pas un ouvrage de vulgarisation sur le BASIC et il n'a pas la prétention de vous apprendre à programmer dans ce langage.

Le **GUEPARD**, vu ses performances, s'adresse en ce domaine à un public déjà averti.

Toutefois, ce guide se veut complet dans la description des possibilités du **SBASIC** en système d'exploitation **NEWDOS . 80** . Version 2 Ø G.

L'auteur est à votre service et attend vos remarques et suggestions pour l'améliorer encore.

Il se présente sous la forme d'un classeur tout justement pour être perfectible dans le cadre du **CLUB GUEPARD**.

Chacune de nos extensions accessible depuis le **SBASIC** est accompagnée d'une nouvelle disquette et de nouveaux chapitres à insérer dans ce manuel.

La lecture préalable du Manuel **NEWDOS** est fortement conseillée pour bien tirer parti du **SBASIC** et du **GUEPARD**.

L'auteur vous souhaite beaucoup de satisfaction.

R.N.

1 - INTRODUCTION AU SBASIC

Le SBASIC disponible sous NEWDOS 80 - 2 0' G est la synthèse du BASIC DISQUE disponible sur le NEWDOS 80 - version 2 et mis au point par Apparat Inc et du Complément de traitement indispensable, compatible avec le BASIC niveau 2 de chez MICROSOFT, mis au point par HBN COMPUTEUR.

Ce BASIC se charge entièrement depuis le DISQUE et laisse à l'utilisateur plus de 38200 octets sur la version de base (64 KO RAM).

La plupart des adresses d'entrée, des pointeurs et l'ensemble des commandes, sont, pour des raisons évidentes de compatibilité avec le BASIC DISQUE et le NEWDOS 80, conservées par rapport au BASIC 2.

Le SBASIC est complémenté d'un certain nombre de commandes nouvelles assurant la gestion vidéo du GUEPARD, la gestion du générateur de son, d'un éditeur pleine page.

Sa gestion nouvelle permet l'arrêt du programme, d'un listing, par appui de la touche BREAK et de le relancer par appui sur une touche quelconque. Elle a permis une accélération du fonctionnement général (boucle de 1 à 10000 en 8 sec).

Dans ce Manuel, chacune des commandes est présentée avec sa syntaxe, une courte explication de son fonctionnement, et un exemple d'emploi significatif.

Ces commandes sont classées par rubrique d'emploi et un INDEX ALPHABETIQUE en appendice B permet de vous y reporter rapidement

En BASIC il existe 2 modes de fonctionnement :

Le mode DIRECT : Le curseur clignote sous OK . à côté de >■

Vous entrez la commande et elle s'exécute immédiatement.

Certaines commandes sont illégales en mode DIRECT (ex. INPUT).

Le mode PROGRAMME :

Un programme en BASIC se compose d'un certain nombre de lignes numérotées par un entier compris entre 0 et 65529.

Un n° de ligne débute toujours par une commande ou une déclaration de variables. Chaque séquence d'instruction dans une même ligne est séparée par : (les 2 points).

Vous entrez en mode programme par la commande RUN.

Certaines commandes sont illégales en mode PROGRAMME (ex. RENUM).

On peut stopper le déroulement du mode PROGRAMME par un appui sur BREAK . Il repart par appui sur n'importe quel touche. SHIFT BREAK l'arrête par retour au mode DIRECT (CONT pour poursuivre).

On peut rendre la touche BREAK inefficace (voir DOS section 2.6)

Chaque erreur dans le déroulement du programme provoque l'arrêt de celui-ci et l'affichage en clair de l'erreur BASIC et éventuellement celle du DOS (sauf en cas de traitement d'erreur voir section 10).

Ce BASIC n'accepte pas un n° de ligne seul (sans commande ou déclaration de variable) sauf en mode AUTO. Il provoque une erreur de syntaxe. Pour supprimer une ligne, utilisez DELETE (ou Dxxxxx).

Dans le descriptif des commandes, les parenthèses encadrent des options et ne doivent pas être reproduites dans la commande elle-même.

2 - CHARGEMENT DEPUIS LE DOS

Le BASIC n'étant pas résident sur mémoire morte (ROM) il faut le charger depuis le DISQUE.

La commande peut prendre différentes formes :

BASIC
BASIC *
BASIC (,n) (,m) (,cde)

Rappel : Les paramètres entre parenthèses sont facultatifs et les parenthèses elles-mêmes à proscrire.

n représente le nombre de zones tampons (Buffers) de 256 octets à réserver pour la gestion des fichiers. Par défaut $n = 3$. Au maximum $n = 15$. L'utilisation de fichiers à accès direct dont la longueur logique d'enregistrement serait variable ($0 < LRL < 256$) nécessite 2 zones pour un même fichier. Dans ce cas l'option V sera spécifié dans n (ex $n = 3 V$).

m représente la plus haute valeur de mémoire utilisable par le BASIC, plus un octet. A défaut le HIMEM DOS est pris en compte. Cette réservation mémoire au-delà de m est utile pour y charger éventuellement des sous-routines en langage machine et donc éviter qu'elles soient écrasées.

cde représente une ligne de texte BASIC, pouvant contenir une ou plusieurs commande (s) BASIC. Elle sera considérée comme une entrée directe au clavier et s'exécutera après la phase d'initialisation du BASIC.

La ligne de commande totale ne peut excéder 80 caractères (32 en mode AUTO) (voir MANUEL NEWDOS).

Si le DOS est en mode Exécution Forcée (option AB du SYSTEME) la commande doit contenir un RUN ou un LOAD option R. (voir section 4.7).

BASIC * permet depuis le DOS de récupérer un programme BASIC en réinitialisant avec les anciennes valeurs de **n** et **m**. Ceci à condition que le retour au DOS se soit fait dans de bonnes conditions (RESET ou CMD'S, ...) et qu'aucune manœuvre ne soit venue écraser la zone Texte BASIC. Si cette fonction est réalisable, elle forcera la commande LIST après la réinitialisation. Elle vous renverra au NEWDOS READY dans le cas contraire.

Exemples :

BASIC charge le BASIC, réserve 3 zones tampons fichiers, avec pour HIMEM celui du DOS et affiche OK dans l'attente de la prochaine commande.

BASIC, CLEAR 3000 : A = 1 : RUN "XXX", V
charge le BASIC, réserve 3 zones tampons, 3000 octets pour les variables de chaîne, attribue 1 à la variable A et charge le programme XXX depuis le disque et débute son exécution sans remettre les variables à zéro (suffixe V) (voir section 4.8).

3 - LES SYMBOLES, CONSTANTES ET VARIABLES.

3.1 - LES SYMBOLES DE BASE

Comme tous les langages évolués, le Basic est construit à partir d'un ensemble de symboles de base. Cet ensemble est simple, et varie très peu d'un constructeur à un autre. Il est composé :

- de chiffres 0, 1, . . . , 9
- de lettres majuscules A, B, . . . , Z
- d'opérateur arithmétiques
 - addition +
 - soustraction -
 - multiplication X
 - division /
 - exponentiation ↑
- des parenthèses gauche et droite ()
- d'opérateurs relationnels
 - égal =
 - inférieur à <
 - supérieur à >
 - inférieur ou égal à <= ou =<
 - supérieur ou égal à >= ou =>
 - différent de <> ou ><
- d'opérateurs logiques
 - opérateur "ET" AND
 - opérateur "OU" OR
 - opérateur "NON" NOT
- de caractères spéciaux
 - guillemet "
 - dollar \$
 - dièse #
 - point d'exclamation !
 - pourcent %
 - et commercial ou perluète &
 - apostrophe '
 - point d'interrogation ?
 - point .
 - deux points :
 - virgule ,
 - point virgule ;
 - blanc ou espace
 - à pour le a commercial à

Tout le langage Basic est construit à partir de ces symboles et comporte essentiellement des notions :

- de constantes,
- de variables,
- de fonctions,
- de mots réservés pour définir les instructions.

3.2 - LES CONSTANTES ET VARIABLES

Dans un programme BASIC interprété, on est amené à traiter de constantes et de variables, et parmi celles-ci, deux catégories :

- constantes et variables numériques,
- constantes et variables de chaîne (alphanumériques).

3.2.1 - LES CONSTANTES NUMERIQUES

Elles sont traitées en virgule flottante et jusqu'à 16 chiffres significatifs. Le signe + est facultatif pour une constante positive, le signe - est impératif pour un nombre négatif.

Elles peuvent être entière : - 4327
 décimale : 7.235022

La prise en compte de cette constante par l'interpréteur est d'autant plus longue que la constante possède de chiffres significatifs. A chaque fois que cette constante doit être remémorisée plusieurs fois il y a gros intérêt à la déclarer comme variable (à l'intérieur d'une boucle par exemple).

Exemples :

```
10 Z = 3.14116
20 FOR I = 5 TO 100
30 X = I * Z ↑ 2
40 NEXT
```

Attention, pour une question de standardisation nous avons conservé la notation décimale US. Le point correspond à la virgule décimale et vice et versa.

Notation : $N E + x = N \times 10^x$ en simple précision.

Par exemple $2.34 E + 4 = 23400$

Notation : $ND + x = N \times 10^x$ en double précision.

Par exemple $1.234567890123456 D + 17 = 1234567890123456\emptyset$

3.2.2 - LES CONSTANTES DE CHAÎNE

Une constante de chaîne est une suite de caractères alphanumériques délimitée par des guillemets.

Ex : "BONJOUR" "123PERDU"

" " est une constante de chaîne nulle, sa longueur maximum est de 255 caractères.

3.2.3 - LES VARIABLES SIMPLES

Une variable est identifiée par les 2 premiers caractères de son nom et le premier doit être alphabétique.

Ex : AB , C , F6 , COeff ,

(attention COEFF et COTOT identifie la même variable CO).

Ne jamais employer de mots réservés (commande BASIC ...) comme nom de variable. THEN = 12 est à proscrire et provoquera une erreur.

Un attribut, à la suite du nom, permet de classer les variables en :

— variables entières	%	AB%
— variables simple précision	!	COL !
— variables double précision	#	FIN #
— variables de chaîne	\$	KZ \$

Sans attribut, elle sera interprétée comme variable simple précision : COL = COL !

Pour s'éviter ces attributs, on peut prédéfinir d'avance la catégorie de variables, ceci afin d'alléger l'écriture et le traitement en employant l'instruction DEFxxx (voir section 5.3).

3.2.4 - LES VARIABLES INDICEES

Pour faciliter certains traitements à caractère répétitif, pour utiliser les variables en boucles, ou conditionnelles, il est souvent intéressant d'utiliser des variables indicées

Ex : M (2) , MJ% (36) , ABC (5,3,1)

L'ensemble des éléments d'une variable indicée se regroupe en un tableau à n dimensions, n étant le nombre d'indices séparés par des virgules à l'intérieur des parenthèses.

AB (1,2,4) est un élément d'un tableau à 3 dimensions.

L'indice peut être une variable et elle-même indicée.

A (I) , B (I (J))

Cet indice est toujours un entier, égal ou supérieur à 0.

3.2.5 - AFFECTATION DES VARIABLES

Le symbole d'affectation est le caractère =

Ex : A = 36.27 la variable simple précision A prend la valeur 36.2700 (6 chiffres significatifs).

B = A + C la variable simple précision B prendra la valeur de l'expression A + C (somme des 2 variables).

C \$ = "TOTO" la variable chaîne C \$ prend la valeur TOTO (les guillemets ne font que délimiter).

D % = E # la variable entière D% prend la valeur tronquée de la variable double précision E #

D = E # la variable simple précision D prend la valeur arrondie de la variable double précision E #

Attention, une variable numérique ne peut être affectée à une variable de chaîne.

3.2.6 - EVALUATION DES EXPRESSIONS

Une variable peut se voir affecter une expression plus ou moins complexe, composée de constantes, de variables, d'opérateurs et de fonctions (sur chaîne ou mathématiques).

Ex : $A = (A + \cos(X) \uparrow Z) / A - \cos(X)$
 $A \$ = \text{RIGHT\$}(" \text{BONJOUR MADAME}", 7)$

Les formes d'expression peuvent être fort complexes.

L'interpréteur BASIC effectue son évaluation en fonction de priorité sur les opérateurs.

- | | |
|---|-----------------------------------|
| 1 | Exponentiation |
| 2 | — Négation |
| 3 | Multipliation et / division. |
| 4 | + Addition et — soustraction |
| 5 | NOT, AND, OR Opérations logiques. |

L'interpréteur BASIC calcule chaque sous expression entre parenthèses en commençant par la plus intérieure et à profondeur égale il traitera la plus à gauche en premier.

Soyez clair dans vos expressions et utilisez soigneusement les parenthèses.

Ex : $X = A + D * \sin(A) - (4 - 2 \uparrow 5) / A$

L'interpréteur calcule dans l'ordre $2 \uparrow 5$ puis ajoute 4 et divise par A.
calcule le $\sin(A)$, multiplié par D
effectue enfin $A +$ résultat 2 — résultat 1.

4 - LES COMMANDES GENERALES

REPERTOIRE DES COMMANDES GENERALES DU SBASIC :

4-1. AUTO	4-10. CMD	4-18. CMD "C"
4-2. DELETE	4-11. MERGE	4-19. CMD "E"
4-3. DI, DU	4-12. NAME	4-20. CMD "F"
4-4. LIST, LLIST	4-13. KILL	4-21. CMD "O"
4-5. EDIT	4-14. RENUM	4-22. CMD "J"
4-6. SAVE	4-15. REF	4-23. CMD "T"
4-7. LOAD	4-16. NEW/RENEW	4-24. CMD "R"
4-8. RUN	4-17. TRON/TROFF	4-25. CMD "S"
4-9. CONT		

4.1 - AUTO

AUTO (aaaa) (, inc)
A (aaaa) (, inc)

Cette commande permet la génération automatique des numéros de ligne. Si aucun paramètre n'est spécifié, le départ se fera à 10 avec un incrément de 10. Dans le cas contraire, le départ se fera à aaaa (Ø par défaut) avec l'incrément spécifié (à défaut 10). On peut ainsi entrer à la suite les commandes désirées.

A est équivalent à AUTO

Chaque appui sur "ENTER" fait passer à la ligne suivante.

Chaque n° de ligne vide n'est pas pris en compte.

Un appui sur BREAK fait quitter le mode AUTO.

Si une ligne devait déjà être utilisée, son numéro serait suivi d'un X . L'appui sur BREAK permet de quitter le mode AUTO en laissant cette ligne intacte. Un appui sur ENTER la remplace ou l'élimine.

Exemples :

AUTO	génère les lignes 10, 20, 30, ...
AUTO 4, 4	génère les lignes 4, 8, 12, ...
AUTO , 5	génère les lignes Ø, 5, 10, ...

4.2 - DELETE

DELETE aaaa (- bbbb) ou D aaaa (- bbbb)

Cette commande permet d'effacer la ligne aaaa ou si bbbb est précisé, toutes les lignes comprises entre aaaa et bbbb.

D est équivalent à DELETE

Exemples :

DELETE 5 efface la ligne 5.
D 100-200 efface les lignes entre 100 et 200 inclus.

4.3 - DI, DU

DI aaaa,bbbb DI . ,bbbb
DU aaaa,bbbb DU . ,bbbb

aaaa est le n° de ligne à transférer ou reproduire. Par défaut le point vise la ligne courante (la dernière listée ou éditée).

bbbb est le n° de ligne récepteur.

DI transfère la ligne aaaa en bbbb (en effaçant aaaa).

DU reproduit la ligne aaaa en bbbb (sans effacer aaaa).

Exemple :

DU 110, 2000 reproduit la ligne 110 en 2000.

4.4 - LIST, LLIST

LIST (aaaa) (- bbbb) LLIST (aaaa) (- bbbb)
L (aaaa) (- bbbb)

Cette commande permet d'afficher sur l'écran (LIST) ou sur l'imprimante (LLIST) les lignes de programme BASIC à partir de la ligne aaaa (la première par défaut) jusqu'à bbbb (la dernière par défaut).

L est équivalent à LIST.

Un appui sur BREAK permet l'arrêt du défilement et un appui sur une touche quelconque de le reprendre.

Un appui sur SHIFT et BREAK stoppe le listing en cours.

Ce BASIC possède un certain nombre de possibilités intéressantes au niveau de la fonction LIST : il accepte les commandes directes suivantes :

.	(point)		liste la ligne courante.
			liste la ligne suivante.
			liste la ligne précédente.
;	ou SHIFT		liste la première ligne.
/	ou SHIFT		liste la dernière ligne.
:			liste une page complète, la ligne courante en bas, celle du haut devenant la ligne courante
à			liste une page complète, la ligne courante en haut, celle du bas devenant la ligne courante
,			édite la ligne courante.

Pour être pris en compte, le caractère doit être le premier frappé sur la ligne (l'usage de , impose l'appui sur "ENTER" avant une nouvelle entrée efficace).

4.5 - EDIT

EDIT (aaaa) (- bbbb) E (aaaa) (- bbbb)

Cette commande permet de passer en mode éditeur pleine page et de modifier ainsi à loisir des lignes de texte avec un outil puissant et facile à utiliser (reportez-vous à la section 11).

4.6 - SAVE

SAVE "Carfich 1" (, A)

Cette commande permet de sauvegarder sur le disque le programme présent en mémoire central dans le Buffer de texte.

Carfich 1 désigne les caractéristiques du fichier de sauvegarde (voir Manuel NEWDOS 80 page MN 2.1).

Si Carfich 1 existe déjà, il est écrasé pour être remplacé par le nouveau

Sans option, le programme est sauvegardé en format compressé.

Avec l'option **A**, le programme est sauvegardé en format ASCII.

Exemple : SAVE "NOMPRO : 1 " sauvegarde le programme résident sous le nom de NOMPRO sur le lecteur 1 en format compressé.

4.7 - LOAD

LOAD " Carfich 1 " (, R) (, V)

Cette commande permet de charger en mémoire centrale un programme sauvegardé sur disque.

L'option **R** permet de lancer le programme automatiquement (elle est équivalente à RUN "Carfich 1", voir section 4.8).

Un chargement d'un nouveau programme efface bien sûr le précédent mais surtout détruit les variables et ferme les fichiers. Un chaînage de sous-programmes traitant des mêmes fichiers et variables peut s'avérer indispensable (gestion de l'espace mémoire). L'option **V** permet de sauvegarder variables et fichiers pendant le chargement et le lancement du programme qui est automatique.

Exemple :

LOAD "NOMPRO" , V : le programme NOMPRO se charge depuis les disquettes montées (recherche à partir du lecteur 0), et se lance automatiquement en préservant variables et fichiers.

4.8 - RUN

RUN (aaaa)
RUN "Carfich 1" (, R) (, V)

Cette commande permet le lancement d'un programme, donc l'exécution de chacune des lignes qui le compose.

RUN seul lance le programme contenu en mémoire centrale à son premier n° de ligne.

RUN aaaa lance le programme à compter de la ligne aaaa.

RUN "Carfich 1" charge le programme Carfich 1 et démarre son exécution.

Avec l'option **R**, les fichiers éventuellement ouverts restent ouverts. Avec l'option **V**, les fichiers et les variables sont préservées (seul les fonctions FN son perdues).

En effet, sans ces options, la commande RUN ferme les fichiers et efface toutes les variables.

Exemple :

RUN 1000 lance le programme résident à la ligne 1000.

4.9 - CONT

CONT (mode direct uniquement)

Cette commande permet de poursuivre l'exécution d'un programme interrompu par un SHIFT BREAK ou la commande STOP.

Si au cours d'une interruption de programme, il y a création, annulation ou modification d'une ligne, les variables sont écrasées et la fonction CONT impossible. Il faudra repartir à zéro.

4.10 - CMD

CMD "Comdos"

Cette commande permet l'exécution, depuis le BASIC, de commandes spécifiques au DOS y compris le lancement de programmes en langage machine. Il faut toutefois préserver l'espace mémoire réservé au BASIC et ne pas créer d'interférence, car dans ce cas, le retour au BASIC à l'issue de l'exécution n'est plus garanti et se fera au NEWDOS READY.

Cette commande permet également de traiter des fonctions et des problèmes spéciaux (voir section 4-18 et suivantes).

Pour mémoire, CMD"S provoque un retour au NEWDOS READY.

Exemple :

10. CMD "BREAK, N" : l'exécution de la ligne 10 provoquera l'exécution de la commande DOS BREAK, N ce qui aura pour effet de rendre la touche BREAK inefficace sous BASIC et le programme se poursuivra sans pouvoir être interrompu.

4.11 - MERGE

MERGE "Carfich 1"

Cette commande permet la fusion du programme résident avec celui sauvegardé sous Carfich 1.

MERGE peut s'utiliser en mode direct ou mode programme et dans ce cas :

- ne doit pas être une partie de définition de fonction (DEFN)
- ne doit pas faire partie d'une sous-routine (GOSUB - RETURN)
- ne doit pas faire partir d'une boucle (FOR - NEXT)
- doit être la dernière commande d'une ligne.

Le fichier Carfich 1 ne peut contenir les mêmes numéros de ligne que le programme résident (si c'est le cas, utilisez CMD "F", DELETE, voir section 4.20).

Le programme se poursuivra à la ligne suivant la commande MERGE qui doit faire partie du programme résident.

MERGE conserve toutes les variables, mais ferme les fichiers.

Exemple :

MERGE "SUITE / BAS : 1"

4.12 - NAME

Cette commande est illégale sous SBASIC

Utilisez CMD "RENAME ..."

4.13 - KILL

KILL "Carfich 1"

Cette commande détruit le fichier Carfich 1 sur le disque.

Exemple :

KILL "DOMMAGE / TXT : 1"

4.14 - RENUM

Uniquement en mode DIRECT

RENUM (dddd) (, inc) (, aaaa) (, bbbb) (, U) (, X)

Cette commande permet de renuméroter tout ou partir d'un programme résident en mémoire centrale. Toutes les références ayant trait aux lignes modifiées seront mises à jour pour garder, bien sur, la cohésion du programme.

L'option **U** permet, sans effectuer l'opération, de vérifier qu'il n'y a aucune erreur risquant de rendre la numérotation impossible, voir risquée. (n° de ligne indéfinie, erreur de syntaxe dans des lignes utilisant les n° de ligne ...)

RENUM **U** est une commande parfaite pour vérifier qu'il n'y a pas eu de saut à une ligne indéfinie sur l'ensemble d'un programme.

L'option **X** permet de ne pas compter comme erreur un n° de ligne indéfinie en dehors de la zone à renuméroter (cette option permet à l'utilisateur de laisser volontairement ces lignes ainsi pour emploi futur comme sous MERGE ...)

aaaa représente le premier numéro de ligne à modifier (Ø par défaut).

bbbb représente le dernier numéro de ligne à modifier (65529 par défaut) et doit être supérieur à aaaa.

dddd représente la nouvelle position souhaitée pour la première ligne. (10 par défaut).

inc représente l'incrément de renumérotation (10 par défaut).

Les nouvelles lignes ainsi générées ne doivent pas empiéter sur d'autres lignes de texte sauf sur celles à transférer et ne doivent pas bien sur sortir du cadre (Ø - 65529).

Au moins un des paramètres doit être spécifié, le minimum étant la virgule.

RENUM, renumérote tout le programme à partir de 10 avec incrément de 10.

Si un paramètre fait défaut et qu'il n'est pas le dernier, son emplacement doit lui être réservé entre 2 virgules.

Si une erreur est détectée pendant l'exécution de la commande, le texte n'est pas altéré et les lignes fautives sont affichées avec :

xxxx / U xxxx est une ligne indéfinie.
xxxx / X la ligne xxxx présente une erreur de syntaxe.
xxxx / S la ligne xxx contient un mauvais n° de ligne.

La fonction RENUM refuse de prendre en compte la ligne Ø (utilisez DU ...)

Si une erreur fatale devait se produire à l'exécution (pour l'éviter, utilisez d'abord l'option U), la sortie se ferait en DOS READY sans retour possible au programme (BASIC ✕ inutilisable).

SYS 11 / SYS et SYS 13 / SYS doivent être présent lecteur Ø pour exécuter la commande RENUM.

Exemple :

RENUM, 2000, , 3000 , 4000 : Les lignes comprises entre 3000 et 4000 seront renumérotées, à partir de 2000 avec un incrément de 10.

4.15 - REF

uniquement en mode direct.

REF (X) (\$) (nn) (aaaa)
REF (= xxxx) (" xxxx) (à aaaa)

Cette commande, puissante aide à la programmation, recherche et trouve à travers le programme résidant en mémoire centrale, un n° de ligne, un entier, une variable, une chaîne de caractères, une commande BASIC.

L'option X affiche à l'écran et exclu \$ qui affiche sur l'imprimante.

nn représente la variable à trouver sans son attribut et les deux premières lettres significatives seulement.

aaaa représente un n° de ligne dont il faut trouver tous les appels.

REF X affiche à l'écran toutes les variables, tous les entiers et n° de ligne.

REF \$ idem sur l'imprimante.

REF nn affiche toutes les références ayant trait à la variable nn

REF aaaa affiche toutes les références ayant trait au n° de ligne aaaa.

REF affiche la prochaine ligne complète ayant un rapport avec la dernière demande REF nn ou REF aaaa, jusqu'à la fin du texte et recommence ensuite au départ.

REF = xxxx permet de rechercher la séquence xxxx, sous sa forme compressée s'il y a lieu (TOKEN) à travers tous le programme y compris à l'intérieur des chaînes de caractères.

REF " xxxx permet de rechercher la séquence xxxx, mais sous sa forme littérale, éliminant ainsi les commandes BASIC.

REF à aaaa exécute REF mais à partir de la ligne aaaa seulement.

Si la séquence xxxx dépasse 16 caractères, seuls les 16 premiers participent aux recherches.

Durant l'exécution de la commande REF, BREAK permet une pause, ENTER pour continuer,  pour sortir de la commande.

SYS 12 / SYS doit être présent lecteur Ø pour permettre l'exécution de REF.

Nota : REF X nn ou REF X aaaa ou REF \$ nn ou REF \$ aaaa s'exécute comme REF X ou REF \$, mais à partir de nn ou aaaa.

Exemple :

REF AB : Liste toutes les lignes traitant de variables portant AB comme 2 premières lettres et en précisant les attributs éventuels.

4.16 - NEW / RENEW

NEW

RENEW

La commande **NEW** "vide" artificiellement le Buffer Texte du programme résident en détruisant le pointeur initial. Il remet toutes les variables à zéro, mais il ne change pas le HIMEM et ne remet pas en cause la dernière fonction CLEAR xxxx.

Si cette commande vous a échappée, il n'est pas trop tard si vous n'avez encore pas créé d'autres lignes, pour réajuster le pointeur et retrouver ainsi votre programme en utilisant la commande **RENEW** (même après un retour au DOS par CMD'S à condition de ne pas tenter un BASIC X dans ce cas là, mais un retour normal et RENEW).

4.17 - TRON / TROFF

TRON

TROFF

Ces commandes gèrent le mode **TRACE**. Le mode TRACE permet durant l'exécution l'affichage du n° de ligne courant et permet donc une éventuelle mise au point.

TRON branche le mode TRACE.

TROFF supprime le mode TRACE.

Le mode pas à pas déclenché par la commande CMD " F = SS " lui est souvent préférable en tant qu'outil de mise au point (voir section 4.20).

4.18 - CMD"C"

CMD"C" (, S) (, R)

CMD"C" supprime, pour le programme résident, tous les espaces inutiles (sauf à l'intérieur des chaînes), toutes les remarques, même si la ligne n'est faite que de commentaires (la ligne est alors supprimée)

CMD"C" , S supprime les espaces sans toucher aux remarques et commentaires

CMD"C" , R supprime les remarques et commentaires sans toucher aux espaces.

Attention, si un appel (par GOTO ou GOSUB) est effectué à une ligne de commentaires et que celle-ci se trouve supprimée, il y a existence d'appel à un n° de ligne indéfinie. Exécuter un RENUM pour confirmer et corriger

CMD«C» n'effectue pas de compression d'espace dans les lignes où cela pourrait générer une erreur de syntaxe à l'exécution et les signale avant de rendre la main.

Exemple :	10 IF F OR D THEN 20
ne peut devenir	10 IFFORDTHEN20Ø

4.19 - CMD"E"

Affiche la dernière erreur DOS rencontrée par le BASIC.

4.20 - CMD''F''

CMD''F'', f_c ou f_c est une fonction spéciale

CMD''F = POPS'' CMD''F = POPR'' CMD''F = POPN''

POPS purge tous les niveaux de RETURN en cours et les niveaux boucle FOR-NEXT, laissant ainsi le programme poursuivre sans niveau de retour.

POPR purge le dernier niveau de RETURN

POPN purge le dernier niveau de FOR-NEXT

CMD''F = POPN'', var purge le niveau de FOR-NEXT de la variable var

CMD''F = SASZ'' , XXX

Cette fonction permet, sans détruire les variables de chaîne, de réajuster en plus ou en moins l'espace qui leur était alloué jusque là (par un CLEAR xxxx ou un autre CMD" F = SASZ", xxxx)

xxxx représente la nouvelle valeur à attribuer.

Une erreur est générée si l'espace devient trop petit pour les variables et trop grand pour le HIMEM. Utilisez alors CMD" F = ERASE ou KEEP"

**CMD" F = ERASE" , var 1, var2,... CMD" F = KEEP" ,
var 1, var2,...**

Cette commande permet de nettoyer l'espace mémoire de façon sélective pour certaines variables. Le nombre de variables est illimité et peut s'exprimer sur plusieurs lignes.

avec ERASE les variables citées sont supprimées

avec KEEP seules les variables citées sont conservées, les autres sont supprimées.

si var x est une variable indicée, l'ensemble du tableau est supprimé, permettant ainsi un nouveau dimensionnement.

On peut donc exprimer les variables sur plusieurs lignes en ignorant les lignes de commentaires éventuels, à condition de terminer la ligne par une virgule indiquant une suite.

```
Exemple : 10 CMD" F = KEEP", HS, BS,  
          20 A%, CO ! ,  
          30 REM  ASTUCIEUX NON ?  
          40 CX$
```

CMD''F = SWAP'' , var1, var2

Cette commande échange les 2 variables var1 et var2 qui doivent être du même type (chaînes, SP, DP, entières)

CMD "F", DELETE aaaa - bbbb

La fonction DELETE en mode programme s'exécute correctement mais retourne la main au mode DIRECT

Cette commande permet de détruire les lignes comprises entre aaaa et bbbb et de poursuivre l'exécution sur la ligne suivant la commande.

Elle conserve les variables sauf les DEFFN des lignes détruites

Elle ne peut faire partie d'une FN, d'une boucle, ou d'une sous routine et doit être la dernière commande de la ligne .

CMD''F = SS'' (, aaaa) (, N)

Cette commande permet d'entrer en mode programme Pas à Pas, à partir de la ligne aaaa si elle est spécifiée (par défaut ce sera la 1ère ligne)

Le n° de la ligne en traitement s'affiche dans le coin supérieur droit de l'écran

Un appui sur ENTER autorise l'exécution de la ligne et l'affichage de la suivante

Un appui sur SHIFT BREAK provoque un retour au mode direct

L'option N permet d'annuler le mode pas à pas.

4.21 - CMD"O"

```
CMD"O" . M , Var1 ( , var2, ...)  
CMD"O" , M , * ivar (Ø), var 1 ( , var 2 ...)
```

Ces 2 commandes permettent des tris de variables en mémoire centrale :

- la première ligne en **Tri direct**
- la seconde ligne en **Tri indirect**

Les variables doivent être indicées et appartenir donc à des tableaux

Il nous faut ici définir la notion de nombre relatif d'éléments (NRE) dans les tableaux à trier.

```
admettons le tableau à 3 dimensions A(R1 , R2 , R3)  
et var1 = A ( X1 , X2 , X3)  
alors le NRE du tri sera égal à  $X1 + X2 * (R1 + 1) + X3 * (R1 + 1) * (R2 + 1)$   
pour un tableau à 1 dimension ,  $NRE = X1$ 
```

Cette définition pour signaler que dans un tri de tableaux multiples, il faut retenir que les NRE de chaque tableau var1, var2, ... doivent être égaux

Le Tri direct s'effectuera en alternant les variables dans les tableaux pour les ranger dans l'ordre ascendant ou descendant (si présence du signe — devant var). var1 en premier et si des éléments de var1 sont égaux , dans l'ordre prescrit pour var2.

Le tri indirect s'effectuera sur ivar (i) qui servira alors d'indice pour la recherche sans affecter les variables var1, var2, ...

n désigne le nombre d'éléments à trier et ne doit pas être supérieur au nombre d'éléments restant à trier dans les tableaux

var1, var2, ... le premier élément du tableau à trier, le signe — affectant l'ordre descendant pour le tri.

ivar (Ø) doit être une variable indicée entière

un maximum de 9 tableaux peuvent être triés ainsi relativement

On peut limiter le tri d'un tableau de variables de chaînes en spécifiant **x** , le **xième** caractère et **y** le nombre de caractères derrière **x** à devoir être pris en compte pour le tri : var2 (i) (x , y)

Le tri s'effectue dans l'ordre (ou l'inverse) des codes ASCII, c'est-à-dire espace, caractères spéciaux, chiffres, lettres, ...

```
Exemple : 10 DIM A$ (50) , B$ (50) , X% (50) , D$ (50) ,  
           C ! (50)  
           20 N = 50  
           30 CMD"O" , N , A$ (1) , B$ (1) (3,2)  
           40 CMD"O" , N , * X% (Ø) , D$ (50) , -C ! (50)
```

La ligne 30 trie en mode direct les tableaux A\$ et dans B\$ les 3ème, 4ème caractère

La ligne 40 trie en mode indirect les tableaux D\$ et dans l'ordre descendant C !

4.22 - CMD"J"

CMD"J", date1, date2

Cette commande permet la conversion de date1 en date2

Si date1 est au format jj/mm/aa, date2 sera au format jjj, le nième jour de l'année

Si date2 est au format -yy/jjj, date2 sera au format jj/mm/aa

Exemple : CMD"J", " -84/345", D\$
alors on obtient : D\$ = " 11/12/84"

4.23 - CMD"T"

Cette commande invalide toutes les routines liées aux interruptions (comme l'affichage de l'heure)

4.24 - CMD"R"

Cette commande revalide toutes les routines liées aux interruptions

4.25 - CMD"S"

CMD"S"

CMD"S = comdos"

Cette commande permet le retour au DOS, avec ou sans exécution d'une commande DOS

Exemple : CMD"S = DIR"

Le BASIC est quitté et le retour à NEWDOS

READY est précédé de l'exécution de la commande DIR.

5 - LES INSTRUCTIONS GENERALES

REPERTOIRE DES INSTRUCTIONS GENERALES DU SBASIC

- 5-1. CLEAR
- 5-2. DIM
- 5-3. DEFxxx
- 5-4. DEF FN, FN
- 5-5. DATA, READ, RESTORE
- 5-6. RND
- 5-7. IF, THEN, ELSE
- 5-8. FOR, TO, STEP, NEXT
- 5-9. GOTO, GOSUB, ON, RETURN
- 5-10. STOP, END
- 5-11. PAUSE
- 5-12. INPUT, LINE INPUT, INKEY\$
- 5-13. DEF USR, USR
- 5-14. INP, OUT, PEEK, POKE
- 5-15. AND, OR, NOT
- 5-16. VARTRP
- 5-17. MEM
- 5-18. FRE
- 5-19. TIME\$
- 5-20. REM

5.1 - CLEAR

CLEAR (xxxxx)

Cette commande remet toutes les variables à zéro

Si xxxxx est spécifié, la zone réservée pour les chaînes sera de xxxxx caractères (à l'initialisation, xxxxx = 50)

Cette réservation est nécessaire si l'on utilise des variables de chaîne dont le nombre de caractères total dépasse 50. Elle doit bien sur se faire en tout début de programme. (voir CMD "F = SASZ", section 4.20).

Exemple : CLEAR 1000, met à zéro toutes les variables et réserve 1000 caractères pour les variables chaînes.

5.2 - DIM

DIM A (x) , B (y), ...

Cette commande permet de dimensionner des tableaux, ce qui est obligatoire si x ou y , ... dépasse 10 et facilite la recherche des éléments (vitesse d'accès)

On peut sans dimensionner, utiliser des tableaux à 3 dimensions dont les indices sont limités à 10, mais on accélère le traitement en les dimensionnant systématiquement.

On ne peut pas redimensionner la même variable, même sur un tableau à dimensions différentes. Si un tableau vous gêne, utilisez CMD "F = KEEP" ou "F = ERASE " (voir section 4.20)

5.3 DEF xxx

```
DEFINT DEFSGN DEFDBL DEFSTR
DEF xxx A (-D)
```

Plutôt que d'utiliser d'encombrants attributs pour caractériser vos variables et pour accélérer le traitement, prenez l'habitude de les déclarer au début de vos programmes.

xxx représente les 4 possibilités

INT pour entier

SGN pour simple précision

DBL pour double précision

STR pour les chaînes

A désigne la lettre (et si — D est spécifié, toutes les lettres comprises entre A et D) qui identifiera ce type de variable si elle débute ainsi.

Exemple : DEFSTR G — M
toutes les variables, commençant par G, H, I, J, K, L et M seront sans autre précision des variables de CHAÎNE.
(il n'empêche que H% restera une variable entière)

5.4 - DEF FN , FN

```
DEFFNmm (X) = f(X) Y = FNmm(X)
```

Cette instruction permet de définir une fonction ayant pour nom FNmm (mm suivant la règle des variables BASIC) et de l'utiliser ensuite directement.

Cette instruction est illégale en mode direct

Exemple : 10 DEF FNEX (Y) = Y/2
20 X = FNEX (4) donnera X = 2

5.5 - DATA , READ, RESTORE

```
- DATA var1, var2, ... , varn
- READ X1, X2, X3, , Xm
- RESTORE
```

L'instruction **DATA**, permet de définir une liste de données de toute nature, séparées par des virgules, pouvant être lues par l'instruction **READ**

L'instruction **READ** permet de placer dans les variables X1, X2, ... Xm les données lues dans l'ordre des DATA

A chaque variable de READ, un pointeur est incrémenté dans la liste des DATA.

A chaque variable de READ, une variable de même attribut doit correspondre et si le pointeur de données a épuisé ses ressources une erreur est déclenchée : DATA EPUISEES

L'instruction **RESTORE** réinitialise le pointeur de données en tête des DATA.

Exemple : 10 DATA 1, 5, BON
20 DATA 15, 8
30 READ A, B, C\$, E, F
40 RESTORE
50 READ G, H

donnera : A = 1, B = 5, C\$ = "BON", E = 15, F = 8, G = 1, H = 5

5.6 - RND

var = RND (x)

L'instruction RND(x) retourne dans var un nombre aléatoire compris entre 0 et x

x doit être inférieur à 32768 et supérieur ou égal à 0

Si $x = 0$ var sera une variable simple précision comprise entre 0 et 1

Si $x \geq 1$ var sera une variable entière comprise entre 1 et x

5.7 - IF THEN ELSE

IF condition (THEN instr1) (ELSE instr2)

Cette instruction permet d'effectuer des tests (**condition**) et s'ils sont ou non positifs, autorise l'exécution de **instr 1** (si oui) ou de **instr2** (si non)

Une des deux conditions de branchement (THEN ou ELSE) au moins doit être spécifiée.

Condition peut contenir plusieurs tests utilisant les opérateurs relationnels et logiques

Une instruction instr1 ou instr2 peut contenir plusieurs instructions séparées par : (les deux points) et se termine, soit en fin de ligne, soit à la condition suivante (ELSE ou THEN)

Plusieurs tests peuvent s'imbriquer les uns dans les autres.

Exemples : 10 IF A = 2 AND B\$ = "OK" THEN B = 10 :
GOTO 2000 ELSE B = 0
Si A = 2 et B\$ = "OK", alors B = 10 et on saute à la ligne 2000 sinon B = 0 et on continue ligne suivante
nota : IF A THEN 200 équivaut à IF A <> 0 THEN 200
ou à IF A = 0 ELSE 200

5.8 - FOR TO STEP NEXT

FOR var = x TO y (STEP z)
NEXT (var)

Cette instruction permet d'effectuer des Boucles de programme

var désigne la variable de contrôle

x désigne la valeur de départ de cette variable

y désigne la valeur finale de cette variable

z désigne l'incrément venant s'ajouter à var à chaque passage (STEP z est facultatif, par défaut l'incrément est de 1)

La boucle est délimitée par l'instruction **NEXT** qui renvoie au **FOR** correspondant à (var) (la plus proche par défaut) tant que var n'est pas $\geq$ à y

L'instruction NEXT doit toujours être précédée du FOR associé

La variable de contrôle est numérique et non indicée

Le branchement de l'extérieur vers l'intérieur d'une boucle est interdit

On peut toujours quitter une boucle avant que var n'ait atteint sa limite. Var conserve alors sa valeur.

Un programme peut contenir plusieurs boucles imbriquées, mais non emboîtées.

Lorsque l'on quitte une boucle, var correspond à la dernière valeur possible augmentée de l'incrément.

```
Exemples : 10 FOR I = 0,5 TO 10 STEP 0.5
            20 PRINT I
            30 NEXT
```

5.9 - GOTO GOSUB ON RETURN

```
(ON exp) GOTO aaaa (, bbbb)...
(ON exp) GOSUB aaaa (, bbbb) ...
RETURN
```

Ces instructions sont des commandes de saut ou de branchement inconditionnels ou conditionnels, calculés ou non.

GOTO aaaa force le déroulement du programme à poursuivre à la ligne aaaa

GOSUB aaaa force le déroulement du programme à se brancher sur le sous-programme débutant à la ligne aaaa et se terminant par un **RETURN**

L'exécution de **RETURN** force le programme à revenir à l'instruction suivant le dernier **GOSUB**

ON x GOTO aaaa, bbbb, ... , nnnn, permet un saut aux lignes aaaa, bbbb, nnnn, en fonction de la valeur de x
si $x = 1$ saut en aaaa
si $x = 2$ saut en bbbb ... etc ...
si $x = \emptyset$ ou $x >$ au nombre de lignes prévues, le programme poursuit à l'instruction suivante.

Mêmes remarques pour **ON x GOSUB ...**

Si x n'est pas un entier, seul la valeur entière de x est prise en considération

On ne doit jamais rencontrer un **RETURN** sans avoir exécuté un **GOSUB**

Un sous-programme peut s'enchevêtrer dans un autre sous-programme (attention à l'analyse)

Ne jamais sortir d'un sous-programme autrement qu'avec **RETURN** ou en précédant la sortie d'un CMD "FF = POPR" (voir section 4.20)

Un sous-programme peut être appelé par plusieurs **GOSUB**

```
Exemple : Essayez 10 INPUT X
                  20 ON X GOSUB 100, 200, 300
                  30 PRINT " on recommence"
                  40 GOTO 10
                  100 PRINT "ENTREE 1" : RETURN
                  200 PRINT "ENTREE 2" : RETURN
                  300 PRINT "ENTREE 3" : RETURN
```

et donnez à x toutes sortes de valeurs.

5.10 STOP END

STOP
END

Ces instructions marquent l'arrêt du déroulement d'un programme :
Provisoire si **STOP**, CONT permet de reprendre
Définitif si **END**

5.11 - PAUSE

PAUSE Var

Cette instruction exécute une pause dans le déroulement du programme

Elle remplace les habituelles boucles d'attente

Le temps de pause s'exprime par **var** en centièmes de secondes

Exemple : 20 PAUSE 1000
effectue une attente de 10 secondes

essayez : 10 PRINT TIME\$
20 PAUSE 1000
30 PRINT TIME\$

5.12 - INPUT LINE INPUT INKEY\$

INPUT ("Texte" ; Var1, Var2, ...
LINE INPUT ("texte" ; Var\$
Car\$ = INKEY\$

Ces instructions gèrent l'entrée des variables à partir du clavier

INPUT permet l'entrée de tous types de variables, même mélangées.
Var1, Var2, ... désigne la variable à saisir et la nature de cette variable (chaîne, entier, ...) soit par déclaration préalable (DEF xxx) soit par attributs (\$, %, ...)

La saisie doit se faire dans le même ordre, séparées par une virgule entre chaque variable (attention pas de virgule dans une variable de chaîne = confusion. Utilisez **LINE INPUT**)

Le signe ? donne le signal de cette saisie, après affichage du "texte" s'il y a lieu

LINE INPUT permet de saisir une chaîne de caractères sous la variable **var\$**, après affichage éventuel du "texte" et sans ?. Une seule variable à la fois pouvant contenir tous les caractères ASCII. Seul un ENTER termine la saisie (var\$ peut contenir 240 caractères)

Car\$ = INKEY\$

Cette instruction permet de ranger dans **Car\$** un caractère clavier lu en passant. En effet, cette instruction n'attend pas le caractère. Si aucun caractère n'est frappé durant le déroulement de l'instruction, **Car\$** sera nul (vide). De plus cette instruction n'affiche pas le caractère saisi.

Un moyen pour bloquer l'attente si besoin : 10 A\$ = INKEY\$: IF A\$ = "" THEN 10

INPUT ET **LINE INPUT** sont des instructions illégales en mode direct.

Attention, un appui sur ENTER au signal de saisie ? de INPUT, conserve la variable à sa valeur antérieure. Si ce n'est pas volontaire et à éviter, faites précéder vos INPUT Var par var = 0 ou var \$ = ""

```
Exemple : 10 X$ = "" : X = 0
           20 INPUT "TEXTE" : X$, X
           30 IF X$ = "" THEN END
           40 PRINTX ; X$ ; " " ; X * 10
```

à l'appel TEXTE ? répondre par exemple MULTIPLIE PAR 10 , 5

```
Exemple : 10 X$ = INKEY$ : IF X$ = "" THEN 10
           20 X = VAL (X$) : ON X GOTO 100, 200, 300
           30 GOTO 10
```

5.13 - DEFUSR USR

DEFUSRn = addr

A = USRn (A%)

Ces instructions permettent d'utiliser depuis le BASIC des routines en langage machine

DEFUSRn = addr permet en fonction de n (0 à 9) de définir le point d'entrée addr (en décimal ou en hexa) de la routine n

A = USRn (A%) permet de l'utiliser une fois définie (une parmi 10 en fonction de n).

Si la routine doit se servir d'un argument (variable), il doit être un entier compris entre - 32768 et + 32768 représenté ici par A% et la routine récupérera cette variable dans le registre HL après avoir exécuté un CALL 0A7FH

Si la routine doit retourner une valeur au BASIC, elle devra la remettre dans HL puis effectuer un JP 0A9AH, et elle sera récupérée en A

La routine doit se terminer par un **RET** pour revenir au BASIC et poursuivre le programme en cours.

L'emploi de ces instructions est réservé aux programmeurs avertis familiarisés avec le langage machine Z80.

```
Exemple : 10 DEFUSR3 = 32000
           100 X = USR3 (0)
           La ligne 10 définit le point d'entrée de la routine
           USR3 et la ligne 100 déclenche cette routine par un
           CALL 32000. Encore faut-il écrire cette routine en
           langage machine !
```

5.14 - INP OUT PEEK POKE

A = INP (port)
OUT port , A
A = PEEK (addr)
POKE addr , A

Ces instructions gèrent l'accès direct du BASIC vers les adresses mémoires 0 à 65535 ou vers les ports 0 à 255

A = INP (port) retourne dans A la valeur de l'octet lu sur le port (port compris entre 0 et 255)

A = PEEK (addr) retourne dans A la valeur de l'octet lu à l'adresse addr (compris entre 0 et 32768, - 32768 et - 1)

OUT port, A envoie sur le port, l'octet A (0 à 255)

POKE addr, A place à l'adresse addr, l'octet A (0 à 255)

Ici encore, il faut bien savoir ce que l'on fait sous peine de perturber le logiciel en cours et surtout le système d'exploitation : réservé aux professionnels ou aux amateurs éclairés

Exemple : POKE 16350, 34
 A = INP (254)

5.15 - AND OR NOT

Ces instructions sont en fait les opérateurs logiques du BASIC. Ils sont valables en utilisation au sens direct (binaire) ou indirect (et/ou du langage courant)

Usage direct (binaire)

NOT : complémentation logique sur 16 bits	NOT 11 = - 12
AND : ET logique	11 AND 6 = 2
OR : OU logique	11 OR 6 = 15

En mode indirect (et/ou logique) en instruction de test

IF A = 0 AND B = 25 THEN ...
IF CS = "BON" OR "BEAU" THEN ...

5.16 - VARTPR

$X = \text{VARTPR}(\text{var})$ ou $\text{VARTPR}(\text{var})$

Cette instruction retourne l'adresse qui pointe sur la variable **var** (soit direct, soit dans X)

Si la variable est : entière	: X	pointe le LSB
	: X + 1	le MSB
	: X	pointe le LSB
	: X + 1	le MSB 1
simple précision	: X + 2	le MSB 2
	: X + 3	l'exposant
	: X	pointe le LSB
	: X + 1	le MSB1
double précision	:	
	: X + 6	le MSB6
	: X + 7	l'exposant
	: X	pointe la longueur
chaîne	: X + 1	le LSB de l'adresse location
	: X + 2	le MSB de l'adresse location

LSB = Least significant byte : Octet de poids faible.

MSB = Most significant byte : octet de poids supérieur.

MSBn = Octet de point supérieur dans l'ordre croissant.

En effet le Z80 traite adresses et données entières en récupérant l'octet de poids faible d'abord, l'octet de poids fort ensuite. L'adresse réelle 402D H sera rangée en 2D40.

La même philosophie est employée par le BASIC pour rester cohérent

Ici encore, l'emploi de cette instruction reste réservée aux initiés

5.17 - MEM

Cette instruction renvoie dans la variable MEM la valeur du nombre d'octets restant libres en mémoire centrale

Exemple : PRINT MEM affichent l'espace restant

5.18 - FRE

$X = \text{FRE}(A)$
 $Y = \text{FRE}(A\$)$

Cette instruction renvoie dans la variable FRE (var) la valeur du nombre d'octets restant libres pour le type de variable considérée

Si var = numérique (A) alors FRE (A) = MEM

Si var = chaîne (A\$) alors FRE (A\$) = espace restant pour les chaînes

5.19 - TIME\$

Cette instruction retourne dans la variable TIME\$ la date et l'heure de l'horloge temps réel sous la forme
jj/mm/aa hh : mm : ss

Exemple : PRINT TIME\$

5.20 - REM

REM commentaires

Cette instruction, la dernière de la ligne en cas de multi-instructions, permet l'introduction de commentaires qui ne seront pas pris en compte dans le déroulement du programme.

Ils serviront à rendre le listing éventuel, et donc la mise à jour intérieure, plus facile.

L'apostrophe ' est équivalente à REM

```
Exemple : 100 A = Z * R * 3.14 : REM CALCUL DU
                                     PERIMETRE
120 S = 3.14 * R ^ 2 : ' CALCUL DE LA
                        SURFACE.
200 REM SUITE
```


6 - LES INSTRUCTIONS ECRAN/IMPRIMANTE/SON

REPERTOIRE DES INSTRUCTIONS ECRAN/IMPRIMANTE/SON DU SBASIC

6.1 - CAR	6.7 - PRINT, à, USING, TAB
6.2 - FON ECR	6.8 - CLS
6.3 - FON CAR	6.9 - POS
6.4 - CONSH, CONSB	6.10 - SET, RESET, POINT
6.5 - INV, FLA, GRE	6.11 - LPRINT
6.6 - CLV	6.12 - PLAY, SOUND, CLM

6.1 - CAR

Car n

Cette instruction permet de changer la couleur des caractères à venir s'afficher sur l'écran

n doit être compris entre $\emptyset$ et 31 (32 couleurs) (voir manuel d'utilisation)

Exemple : Car2 (les caractères qui suivent seront rouge)

6.2 - FON ECR

FONECR n

Cette instruction permet de changer la couleur du fond d'écran (effacement fin de ligne ou fin de page ou écran total)

n doit être compris entre $\emptyset$ et 31

Exemple : FONECR 5 : CLS

6.3 - FON CAR

FONCAR n

Cette instruction permet de changer la couleur du fond de caractère sur l'ensemble de l'écran

n doit être compris entre $\emptyset$ et 31

Exemple : FONCAR $\emptyset$

nota : ces instructions couleur ne sont réellement intéressantes que sur un écran adapté (moniteur couleur ou TV familiale couleur). Une sortie péritel à l'arrière de l'appareil vous permet d'en profiter.

6.4 - CONSH CONSB

CONSH n
CONSB n

Ces instructions permettent de réserver en haut (CONSH) et en bas (CONSB) n lignes d'écran ou le scrolling ne viendra plus déranger vos titres et remarques ou consignes éventuelles

La somme des 2 réservations doit être comprise entre 0 et 22.

on peut modifier cette réservation à tout moment

on peut aller écrire dans les zones réservées par l'instruction PRINT à mais le retour des zones réservées vers la zone de défilement doit être assuré par le programmeur.

Exemple : 10 CONSH 2
 20 CONSB4
 30 FOR I = TO40 : PRINT "AFFICHAGE" ; I : NEXT

6.5 - INV FLA GRE

Ces instructions permettent depuis le BASIC d'effectuer les inversions vidéo et les demi-tons, tout comme la saisie directe au clavier

Chaque instruction **INV** fait passer la suite de l'affichage en inversé (jusqu'à la prochaine instruction INV)

Chaque instruction **FLA** fait de même pour le Flash et **GRE** pour le dégradé (demi-ton)

Exemple : 10 PRINT "HELLO" : INV : PRINT "BONJOUR" : INV

6.6 - CLV

Cette instruction permet de remettre les paramètres vidéo initiaux c'est-à-dire CONSH et CONSB à 0 (écran total disponible)
FONCAR, FONECR et CAR pour fond bleu écriture blanche
INV, FLA et GRE en écriture normale

Elle n'efface pas l'écran

6.7 - PRINT à USING TAB ?

- PRINT "xxxx"
- PRINT (var1); (var2), (var3)
- PRINT à nnn,
- PRINT USING "Param" ; var
- PRINT USING var\$; var
- PRINT TAB (n) ; var

L'instruction PRINT est l'instruction d'affichage écran par excellence

C'est l'instruction la plus utilisée du BASIC

Ses performances sont telles que sa maîtrise complète demande une longue expérience

Les données à afficher peuvent être du texte (entre guillemets), des variables, des éléments de tableau, des expressions arithmétiques et des éléments de mise en page

Les séparateurs sont le point virgule ; et la virgule ,

Les fonctions possibles sont TAB, à , USING

Le point d'interrogation ? équivaut à l'instruction PRINT (hors d'une chaîne)

PRINT sans paramètre effectue un effacement fin de ligne et positionne le curseur en début de ligne suivante.

PRINT "texte" ou **PRINT var1** affichent le texte entre guillemets ou la variable var et effectuent un effacement fin de ligne et positionnent le curseur en début de ligne suivante.

Le point virgule entre les variables et en fin d'instruction empêche l'effacement fin de ligne et le repositionnement du curseur

La virgule effectue une tabulation de 16 caractères

La fonction **TAB (x)**, permet d'effectuer une tabulation sélective de x pour l'affichage qui suivra

Exemple : **PRINT TAB (10) , "TITI"**

La fonction **à xxxx**, permet d'afficher à partir d'une position donnée de l'écran comptée en décimal du haut gauche au bas droit, ligne par ligne (0 à 1023 en 64 colonnes – 0 à 1919 en 80 colonnes)

Cette fonction ne peut apparaître qu'une fois dans une instruction PRINT et suivre immédiatement la commande . xxxx peut être une variable ou une expression.

Exemple : **PRINT à 800, "TITI"**

La fonction **USING format ; var1 (;var2) (; ...)** permet l'édition d'une ou plusieurs données selon un format prédéterminé.

format peut être soit une constante chaîne (entre guillemets) soit une variable de chaînes, avec les caractères spéciaux suivants :

spécifie la position de chaque chiffre. La variable est cadrée à droite, ou sur le . (point décimal) s'il y a lieu. Les positions non utilisées sont remplacées par des blancs à gauche et des 0 à droite (si décimal)

Exemple : PRINT USING " # # # # . # # # "; 12.27
s'inscrira 1 2 . 2 7 0

S'il y a trop de chiffres pour un nombre de positions données, un % suivi du chiffre à partir de la 1ère position signalera le dépassement sans provoquer d'erreur.

Attention le signe — compte pour un chiffre à part entière. Pensez-y dans vos formats car il s'affichera comme tel.

* * deux astérisques (et pas plus ni moins) en début de format feront que l'excédent gauche est remplacé par des *

Exemple : PRINT USING " * * # # # . # # "; 12.27
s'affichera * * 1 2 . 2 7

\$ un caractère \$, selon sa position (contre les # ou avant les * s'il y a lieu), provoquera l'affichage du \$ contre le 1er chiffre ou en tête du format. Le \$, contrairement aux * ne peut tenir lieu de position possible : il vient en plus.

Exemple : PRINT USING " \$ * * # # . # # "; 123.57
s'affichera \$ * 1 2 3 . 5 7
PRINT USING " * * \$ # # . # # "; 123.57
s'affichera * * \$ 1 2 3 . 5 7
PRINT USING " \$ # # # . # # "; 23.57
s'affichera \$ 2 3 . 5 7

+ Le signe + au début ou à la fin du champ permet l'affichage du signe de la donnée à gauche ou à droite de la donnée.

% % Les signes % encadrant des espaces spécifient le nombre de caractères d'un chaîne à imprimer en partant du début et y compris les signes eux-mêmes.

Exemple : PRINT USING " % % " % " ; " ABCDEFGH
affichera ABCDE

! Le signe ! indique qu'il ne faut afficher que le premier caractère de la chaîne.

Le format peut être complexe, pour plusieurs variables, de type différent et contenir des espaces et autres caractères (différents de ceux faisant office de discriminateur) qui seront affichés tel quel dans le format

Exemple : 10 A\$ = "BONJOUR MADAME"
20 B = 27.23
30 C = 70
40 ? USING "% % HOP ### . ## ###"; A\$;
B ; C
affichera BON HOP 2 7 . 2 3 7 0

Une virgule placée dans le format effectue la séparation par 1000 au format US

Exemple : 10.PRINT USING " # , ### . ## " ; 12623.36
s'affichera 12 , 6 2 3 . 3 6

6.8 - CLS

Cette instruction efface l'écran (en le remplissant d'espaces de la couleur de FONECR) et positionne le curseur en haut à gauche (position $\emptyset$)

6.9 - POS

X = POS ($\emptyset$)

X = POS (1)

L'instruction POS avec ($\emptyset$) comme argument fictif, retourne dans X la position du curseur sur la ligne ($\emptyset$ à 63 ou $\emptyset$ à 80)

L'instruction POS avec (1) comme argument fictif, retourne dans X la position du curseur sur l'écran ($\emptyset$ à 1023 ou $\emptyset$ à 1919)

6.10 - SET RESET POINT

SET (x, y)
RESET (x, y)
POINT (x, y)

Chaque matrice caractère de l'écran fait 8 x 12 points que l'on divise en 6 (par 2 en horizontal et par 3 en vertical), pour en faire des caractères graphiques basse résolution. Ce qui donne 160 x 72 points en mode 80 colonnes et 128 x 48 points en 64 colonnes.

SET (x, y) " allume " le point aux coordonnées x et y de l'écran

RESET (x, y) " éteint " le point aux coordonnées x et y de l'écran

POINT (x, y) est une variable qui prend la valeur -1 si le point x, y est allumé et $\emptyset$ s'il est éteint.

Exemple : 10 SET (40, 20)
20 X = POINT (40, 20)
30 RESET (40, 20)
40 Y = POINT (40, 20)

donne en dehors de l'affichage fugace , X = -1 et Y = $\emptyset$

6.11 - LPRINT

Cette instruction exécute vers l'imprimante les mêmes routines que PRINT vers l'écran, l'option à xxxx exceptée.

Si l'imprimante ne devait pas être correctement connectée (branchée, alimentée, on line, ...) le logiciel assure un balayage de cinq secondes avant d'afficher IMPRIMANTE INDISPONIBLE. A ce stade, on peut réparer l'erreur et le logiciel poursuit ou un BREAK pour retour au mode direct.

6.12 - PLAY SOUND CLM

PLAY c, o, n, v, t
SOUND r, nn
CLM

Ces instructions permettent de gérer le générateur de SON depuis le BASIC

Tout ce qui concerne ce générateur est décrit dans le manuel d'utilisation, section 6

PLAY c, o, n, v, t permet de générer une note de musique

c désigne le canal (1 à 3)
o désigne l'octave (0 à 6)
n désigne la note (1 à 12)
v gère le volume (0 à 15)
t gère la durée de la note en centième de seconde (0 à 32768)
v = 16 met le volume sous contrôle du générateur, ce qui permet de contrôler l'enveloppe du signal avec les registres 11, 12 et 13 que l'on peut piloter par l'instruction SOUND

Exemple : 05 SOUND 7,8
 10 FOR I = 0 to 6
 20 FOR J = 1 to 12
 30 PLAY 1, I, J, 10, 20
 40 NEXT J
 50 NEXT I

Exemple : rajouter à l'exemple précédent :
 05 SOUND 7, 8 : SOUND 11 , 20 : SOUND 13, 16
 30 PLAY 1, I, J, 16, 20

SOUND r, nn permet de programmer directement les registres du générateur

r désigne le registre (0 à 14)
nn désigne la variable (0 à 255)

CLM remet tous les registres du générateur de son à zéro.

7 - LES FONCTIONS MATHÉMATIQUES

REPERTOIRE DES FONCTIONS MATHÉMATIQUES DU SBASIC

7.1 - ABS	7.5 - SQR	7.8 - TAN	7.11 - SGN
7.2 - INT	7.6 - COS	7.9 - ATN	7.12 - &H
7.3 - FIX	7.7 - SIN	7.10 - LOG	7.13 - &O
7.4 - EXP			

7.1 - ABS

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

Cette fonction retourne en Y la valeur absolue de X

Exemple : $\text{ABS}(-62.3)$ donne 62.3

7.2 - INT

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

Cette fonction retourne dans Y la partie entière de X (plus grand entier inférieur ou égal à X)

Exemple : $\text{INT}(-3.5) = -4$
 $\text{INT}(3.5) = 3$

7.3 - FIX

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

Cette fonction retourne dans Y la partie entière de x (troncature)

Pour un nombre positif $\text{FIX}(X) = \text{INT}(x)$

Pour un nombre négatif $\text{FIX}(x) = \text{INT}(x) - 1$

Exemple : $\text{FIX}(-3.5) = -3$
 $\text{FIX}(3.5) = 3$

7.4 - EXP

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

Cette fonction retourne dans Y l'exponentielle de X ($Y = e^X$)

Exemple : $\text{EXP}(1)$ donne 2.71828

7.5 - SQR

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

Cette fonction retourne dans Y la racine carrée de X

Exemple : $\text{SQR}(9)$ donne 3

7.6 - COS

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

Cette fonction retourne dans Y le cosinus de X exprimé en radians

Exemple : COS (45) donne 0.525322

7.7 - SIN

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

Cette fonction retourne dans Y le sinus de X exprimé en radians

Exemple : SIN (45) donne 0.850902

7.8 - TAN

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

Cette fonction retourne dans Y la tangente de X exprimée en radians

Exemple : TAN (45) donne 1.61977

7.9 - ATN

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

Cette fonction retourne dans Y l'arc tangente de X exprimé en radians

Exemple : ATN (45) donne 1.54858

7.10 - LOG

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

Cette fonction retourne dans Y le logarithme népérien (base e) de X

Exemple : LOG (2) donne 0.693147

7.11 - SGN

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

Cette fonction retourne en Y la valeur 1 si X est positif
la valeur - 1 si X est négatif
la valeur 0 si X est nul

7.12 - &H

$$Y = \&H \text{ xxxx}$$

Cette instruction retourne en Y la conversion décimale de la valeur hexadécimale xxxx

Exemple : &H0FFFF donne - 1 (pour 65535)

7.13 - &O

$$Y = \&O \text{ nnn}$$

Cette instruction retourne en Y la conversion décimale de la valeur octale nnn

8 - LES FONCTIONS SUR CHAÎNE DE CARACTÈRES

REPERTOIRE DES FONCTIONS SUR CHAÎNE DE CARACTÈRES DU SBASIC

8.1 - ASC	8.6 - MID\$
8.2 - CHR\$	8.7 - STRING\$
8.3 - LEN (C\$)	8.8 - STR\$
8.4 - LEFT\$	8.9 - VAL
8.5 - RIGHT\$	8.10 - INSTR

8.1 - ASC

$X = \text{ASC}(C\$)$

Cette fonction retourne la valeur décimale du premier caractère de la chaîne C\$

Exemple : ASC ("A") donne 65

8.2 - CHR\$

$C\$ = \text{CHR\$}(X)$

Cette fonction, inverse de la précédente, retourne dans la variable de chaîne C\$ le caractère dont la valeur décimale en code ASCII est X

Exemple : CHR\$ (65) donne "A"

8.3 - LEN (C\$)

$X = \text{LEN}(C\$)$

Cette fonction retourne dans X la longueur de la variable de chaîne C\$ (nombre de caractères)

Exemple : LEN ("PAPA AIME") donne 9 (l'espace compte)

8.4 - LEFT\$

$L\$ = \text{LEFT\$}(C\$, x)$

Cette fonction retourne dans la variable de chaîne L\$ les x caractères de gauche de la variable de chaîne C\$

Exemple : LEFT\$ ("BONJOUR" ,3) donne BON

8.5 - RIGHT\$

$R\$ = \text{RIGHT\$}(C\$, x)$

Cette fonction retourne dans la variable de chaîne R\$ les x caractères de droite de la variable de chaîne C\$

Exemple : RIGHT\$ ("BONJOUR" , 4) donne JOUR

8.6 - MID\$

$\text{MID\$}(\text{C\$}, x, y)$

Cette fonction retourne dans M\$, les y caractères, débutant à la position x, de la variable de chaîne C\$

x doit être différent de 0 et inférieur à LEN (C\$) - Y

Exemple : MID\$ (" BONJOUR" , 4,2) donne JO

8.7 - STRING\$

$\text{C\$} = \text{STRING\$}(x, \text{X\$})$

Cette fonction retourne dans C\$ une chaîne composée de x caractères X\$ (qui doit donc être une variable de chaîne de longueur 1)

Exemple : STRING\$ (4, "U") donne UUUU

8.8 - STR\$

$\text{C\$} = \text{STR\$}(X)$

Cette fonction retourne dans C\$ une chaîne conversion de la valeur numérique X

Exemple : STR\$ (123) donne " 123". Attention au signe, le + est remplacé par un espace.

8.9 - VAL

$X = \text{VAL}(\text{C\$})$

Cette fonction (inverse de STR\$) retourne dans X la valeur numérique de la chaîne C\$

Exemple : VAL ("1234") donne 1234

8.10 - INSTR

$X = \text{INSTR}(\text{C\$}, \text{ST\$})$

Cette fonction retourne dans X la position de départ de la chaîne ST\$, dans la variable de chaîne C\$, la recherche débutant à la nième position (n non précisé équivaut à n = 1)

Exemple : X = INSTR (2, "AU BAL CE SOIR") on trouve X = 5

Si ST\$ n'est pas trouvé dans C\$, alors INSTR = 0

Comme pour les fonctions numériques, ces fonctions peuvent être imbriquées et former des expressions complexes à condition de respecter les règles sur les mélanges de type de variable

Exemple : A\$ = MID\$ (LEFT\$ (A\$, VAL(C\$)), 2,2)

9 - LES FICHIERS EN SBASIC

REPERTOIRE DES COMMANDES

- 9.1 - OPEN
- 9.2 - CLOSE
- 9.3 - PRINT#, INPUT#, LINE INPUT#
- 9.4 - EOF
- 9.5 - FIED, LSET, RSET
- 9.6 - MKI\$, MKS\$, MKD\$, CVI, CVS, CVD, CHR\$, ASC
- 9.7 - LOF
- 9.8 - GET, PUT (FIELD AD)
- 9.9 - GET,PUT,(fichiers,MI,MU,MF,FI,FF)
- 9.10 - LOC

Préambule

Le SBASIC est capable de gérer 7 types de fichiers différents avec peu de commandes.

Le fichier à accès séquentiel, ainsi nommé parce que la lecture des informations stockées, ainsi que l'écriture, ne peuvent se faire que dans l'ordre croissant de la place sur la disquette. On ne peut avoir accès aux données en direct. L'accès séquentiel n'étant pas réservé à ce seul type de fichier, nous appellerons celui-ci le PRINT/INPUT (PI) car il est le seul à utiliser ces instructions d'entrée/sortie

Le fichier à accès direct (ou Random Access) ainsi nommé (chez Microsoft 5.) par opposition au séquentiel, parce que le fichier est segmenté en enregistrements (Record) auxquels on peut avoir accès en direct par un n°. Mais il n'est plus le seul à être à accès direct, de ce fait, nous l'appellerons FIELD AD de par la commande spécifique qui le caractérise par rapport aux autres : FIELD

Ces 2 types de fichier sont strictement équivalents à ceux du BASIC Microsoft 5.

Le SBASIC introduit 5 nouveaux types de fichiers

Le fichier "MU" le plus facile à digérer. Les données sont rangées sur le disque en enregistrements de longueur variable et le logiciel place des marques spécifiques pour séparer les enregistrements (SOR) et annoncer les variables. Il pourvoit de lui-même aux séparateurs. L'accès direct à ce type de fichier, qui reste aussi simple à manipuler que le séquentiel, est possible, ainsi que la mise à jour. Il devrait avoir tendance à supplanter le PI (print/input)

Les fichiers "MF" et "FF"

Le F final identifie des fichiers segmentés en enregistrements de longueur fixe et prédéterminée dans l'OPEN, comme pour le FIELD AD, mais la longueur (limitée à 256 pour le FIELD) peut aller ici jusqu'à 4095 caractères.

Le M initial identifie les fichiers où les variables sont stockées dans les enregistrements avec des marqueurs de repérage (octets de contrôles)

Le F initial identifie les fichiers où les variables sont stockées dans les enregistrements sans marqueur de repérage et l'opérateur doit pourvoir à leur sélection au travers des ISEL'S (voir section 9 - 11)

Les fichiers "MI" et "FI"

Le I final identifie des fichiers non segmentés en enregistrements. Seul, l'utilisateur peut contrôler la position des variables dans le fichier. L'utilisation de ce type de fichier permet une grande souplesse mais ne s'improvise pas et demande une longue expérience.

Les M et F initiaux ont la même signification que pour "MF" et "FF"

Un tableau récapitulatif des structures de ces fichiers est donné en appendice. Reportez-vous y souvent à la lecture de ce chapitre. Il vous aidera à faire votre propre synthèse des possibilités offertes.

Pour bien comprendre la structure et le mode d'utilisation de chacun de ces types de fichiers, il vous faut parcourir lentement les sections qui vont suivre au moins une fois sans chercher à utiliser.

A la suite de quoi, une deuxième lecture, le tableau de l'appendice F sous les yeux, et un peu de pratique sur l'appareil, et tout deviendra clair. En effet, c'est plus compliqué à lire qu'à utiliser.

De nombreux ouvrages traitent des fichiers en BASIC et ils vous apporteront des sujets d'utilisation précis, quelques astuces particulières. Consultez-les.

9.1 - OPEN

OPEN "m", bn, "Carfich 1" (,rf) (lrl)

Cette instruction est indispensable pour valider l'accès à n'importe quel type de fichiers, puisqu'elle est celle qui attribue une (ou plusieurs) zone tampon (buffer) : bn (1 à 15) au fichier Carfich 1

Attention pour bn > 3, il est nécessaire de veiller à le préciser avant le chargement du BASIC qui par défaut autorise la réservation de 3 buffers.

m précise le mode d'accès au fichier, si m =

- I accès autorisé pour entrée du fichier vers la mémoire
Par INPUT # si rf non spécifié (séquentiel)
Par GET si rf est spécifié
Le pointeur est positionné en début de fichier
- O accès autorisé pour sortie de la mémoire vers le disque.
Si le fichier n'existe pas, il est créé sur disque et l'accès se fera
Par PRINT # si rf non spécifié (séquentiel)
Par PUT si rf est spécifié
Le pointeur est positionné au début de fichier et l'EOF est positionné à Ø
- E accès autorisé comme pour O, mais pour un fichier déjà existant que l'on veut étendre. L'EOF est donc laissé à son ancienne valeur.
- R accès autorisé dans les deux sens. Si le fichier n'existe pas il est créé sur le disque. L'accès n'est autorisé que par les instructions GET et PUT (accès direct). Le fichier peut être étendu par une instruction PUT au delà de l'EOF si aucun paramètre DOS ne l'en empêche (voir CREATE MANUEL DOS Section 2-13)
- D autorise et exécute comme l'option R, mais interdit quels que soit les paramètres, l'extension du fichier.

rf ce paramètre spécifie le type de fichier à accès direct. rf peut être choisi parmi 6 options

- "FI" repère de IGEL AD non segmenté en enregistrements exclue lrl
- "FF" repère de IGEL AD à longueur d'enregistrement fixe lrl doit être précisé
- "MI" repère de MARQ AD non segmenté en enregistrement exclue lrl. La mise à jour d'un fichier MI est impossible
- "MU" repère de MARQ AD segmenté en enregistrements de longueurs variables. lrl est optionnel et indique la longueur maxi d'un enregistrement. Un fichier MU peut être mis à jour mais sans changer la taille des premiers enregistrements (troncature à droite ou excédent en ØØ)

"MF" repère de MARQ AD à longueur d'enregistrement fixe. lrl doit être spécifié.

Pas de rf : on retrouve le FIELD AD, le classique RANDOM ACCESS. si m = R ou D
on retrouve le séquentiel classique si m = O, I ou E

lrl longueur d'enregistrement

Si rf n'est pas spécifié et le mode R ou D est choisi, par défaut (LrL) = 256 et peut être précisé entre 1 et 256.
(à condition de l'avoir prévu au chargement du BASIC option V). Exemple : BASIC, 3 V.

Si rf est spécifié, lorsque lrl est requis, il peut être compris entre 0 et 4095

En conclusion, un OPEN sans option rf caractérise les fichiers séquentiels et accès direct bien connus. Seule, l'option rf nous engage sur la voie de l'étude d'une gestion de fichiers plus élaborée.

Exemple : OPEN "O" , 1, " FICH1 : Ø" ouverture fichier séquentiel pour accès vers disque, buffer n° 1.

OPEN "R" , 2, "FICH2 : 1" , 64 ouverture fichier accès direct type FIELD à LRL = 64, buffer 2

OPEN D" , 1, "FICH3 : Ø" , "MF" , 71 ouverture d'un fichier MF à LRL = 71 Buffer 1 à accès direct (D)

OPEN "O" , 1, "FICH4 : 1" , "MI" ouverture d'un fichier MI à écriture séquentielle buffer 1

OPEN "I" , 2, "FICH4 : 1" , "MI" ouverture du même fichier MI pour lecture séquentielle buffer 2

OPEN "D" , 3, "FICH5 : 1" , "FF" , 63 ouverture d'un fichier FF à LRL = 63 pour accès direct buffer 3.

9.2 - CLOSE

CLOSE (bn1) (,bn2)...

A l'ouverture d'un fichier, le FCB de ce fichier (partie du DIRECTORY qui le caractérise, lui et ses secteurs sur le disque) est transposé en mémoire centrale pour mise à jour suivant l'évolution du traitement

Aucune mise à jour du FCB sur le disque n'est faite durant les enregistrements d'accès fichier.

L'instruction **CLOSE** permet la fermeture, sélective ou non de ces fichiers.

Elle remplace le FCB mis à jour dans le DIRECTORY

Elle libère la zone tampon concernée par le (s) **bn(s)** en option

Par défaut de précision sur un **bn**, tous les fichiers ouverts sont fermés

Exemple : CLOSE ferme tous les fichiers
 CLOSE 2,3 ferme les fichiers utilisant les buffers 2 et 3

Attention : toute nouvelle extension ou rétraction d'un fichier impose d'effectuer un CLOSE de mise à jour DIRECTORY avant d'oter la disquette concernée ou de quitter le BASIC
Une lecture seule ne l'impose pas.

En accès direct du type FIELD, utilisez si possible la fonction CREATE du DOS. Elle permet de préouvrir un fichier avec un nombre d'enregistrements déterminés remplis de Ø.

voir également section 9.9. PUT, GET avec fp = &&.

9.3 - PRINT # INPUT # LINE INPUT

Ces instructions sont caractéristiques d'entrées/sorties de fichiers **Séquentiels**.

Elles excluent l'emploi de **rf** dans l'**OPEN**.

PRINT # br, var 1 (; ou , var 2) ...

Après l'ouverture d'un fichier en mode "**O**" ou "**E**", cette instruction permet l'écriture sur disque des variables en séquentiel ;

A partir du début en mode "**O**", après le dernier séparateur en mode "**E**" (extension).

bn désigne le n° du buffer du fichier (voir **OPEN**)

var 1 seul sera suivi d'un retour chariot (CR ou ENTER, ØD H en HEXA).

; **var 2** "collera" var 2 à var 1 sans séparateur après une variable de chaîne, avec espace après variable numérique.

, **var 2** placera var 2 avec tabulation d'espace, sans séparateur après une variable de chaîne.

La dernière variable de la ligne sera suivi du CR (séparateur).

Vous pouvez vous-même y placer des séparateurs sous la forme **CHR\$ (13)**.

Les séparateurs en séquentiel jouent un rôle important à la relecture.

Ces séparateurs sont la virgule, le CR (ENTER), le saut de ligne (line-feed) et l'espace après une variable numérique.

La lecture d'un enregistrement se fait par variable, entre chaque séparateur, sauf entre deux guillemets (**CHR\$ (34)**).

Attention donc, une virgule dans une chaîne de caractère sera interprétée comme séparateur sur disque.

Exemple : 10 A \$ = "BONJOUR, PAPA" :
 B \$ = "COMMENT" : C \$ = "VAS-TU"
20 PRINT # 1, A \$
 donnera BONJOUR dans la 1ère variable appelée
 par INPUT
30 PRINT # 1, B \$; CHR\$ (13) ; C \$
 et "PAPA" dans la 2ème et "COMMENT" dans
 la 3ème.

Essayer de ne pas mettre **CHR\$ (13)** pour voir !

INPUT #bn, var 1 (, var 2) ...

La lecture se fera dans l'ordre de l'écriture après ouverture du fichier en mode "I".

La variable récupérée sera celle contenue entre 2 séparateurs (voir remarque sur PRINT #).

Chaque entrée de variable incrémente le pointeur sur la prochaine.

Une demande au delà de la fin de fichier (EOF) provoquera une erreur

La surveillance (par programme) de l'EOF (voir section 9.4) permet d'éviter cette fâcheuse issue.

```
Exemple : 10 OPEN "I", 1, "FICH : 1"
           20 IF EOF (1) = - 1 THEN END
           30 INPUT # 1, A$ : ? A$
           40 GOTO 20
```

LINE INPUT # bn, var 1 \$, (var 2 \$)

permet la lecture de variables de chaîne sans tenir compte d'autre séparateur que le CR (ENTER) (oublie donc les fâcheuses virgules).

Elle s'arrête toutefois à 255 caractères (à défaut de CR).

ATTENTION aux séparateurs en chaîne et en numérique, le CR (ENTER) doit toujours terminer une chaîne à l'enregistrement pour plus de sécurité. En numérique, l'espace généré par le ; suffit, un conseil : ne mélangez pas les 2 !

9.4 - EOF

$X = \text{EOF}(\text{bn})$

Cette instruction retourne dans X la valeur - 1 si le pointeur de lecture fichier en séquentiel est sur la fin du fichier (EOF).

Cette instruction retourne 0 dans X si cela n'est pas le cas.

Cette instruction permet donc de surveiller les entrées fichier pour éviter toute erreur d'appel variable après l'EOF (voir section 9-3).

bn est le n° de buffer attribué au fichier concerné par l'OPEN.

9.5 - FIELD LSET RSET

Dans les fichiers à classement de variables par FIELD dit accès direct (ou Random Accès), le fichier est segmenté en enregistrements (Record) de même longueur (LrL) définie à l'ouverture du fichier dans l'OPEN.

Il faut respecter cette longueur dans tous les OPEN's ultérieurs concernant ce fichier.

Si LrL n'est pas spécifié, cette longueur sera de 256 caractères. Si LrL est spécifié, il doit être compris entre 1 et 256 et le BASIC doit être chargé avec l'option V suivant le nombre de buffer à réserver.

Dans cet enregistrement de longueur donnée, il faut y ranger les variables et c'est là le but des instructions FIELD, LSET, RSET.

AS

FIELD bn, L1A\$ VAR1\$ (, L2A\$ VAR2\$) ... (LnAS VARN\$)

Chaque FIELD effectué pour un même fichier, décompose à priori l'enregistrement en sous-segment de longueur L1, L2...Ln, destinés à recevoir les variables de chaîne var 1 \$, var 2 \$, ... var n \$.

Le fichier est cité de part son numéro de buffer : bn.

Ensuite les instructions LSET et RSET effectueront le rangement dans le buffer de ces sous-segments avec la variable concernée avec ou sans conversion (numérique en chaîne) par les fonctions MKI\$, MKS\$, MKD\$ et CHR\$.

On peut définir plusieurs FIELD pour un même buffer en changeant bien sûr le nom des variables.

Il faut bien sûr que $L1 + L2 + \dots + Ln$ ne dépasse pas la longueur de l'enregistrement lrl.

L1, L2, ... Ln sont à déterminer par l'opérateur en fonction de la variable à ranger, à convenance pour les chaînes, suivant le mode de conversion pour les variables numériques (voir section 9.6).

LSET Varn\$ = Conversion (Var) ou var \$
RSET Varn\$ = Conversion (Var) ou var \$

Ces instructions de rangement travaillent de la même façon :

LSET range après conversion de variable numérique ou en direct pour les chaînes en remplissant depuis la gauche du sous-segment délimité par Varn \$ dans le FIELD et troncuture la variable à droite si Ln est insuffisant ou remplit de blanc l'espace restant.

RSET fait de même à droite et donc troncuture ou remplit de blanc à gauche.

9.6 - MKI\$ MKS\$ MKD\$ CVI CVS CVD CHR\$ ASC

Ces instructions, employées en opérateur de fonction de rangement LSET ou RSET, effectue la conversion en variables de chaîne des variables numériques et vice et versa à la relecture du fichier.

Conversion numérique vers chaînes pour rangement.

Exemple : LSET (ou RSET) Varn \$ = MK x \$ (var) ou
CHR\$ (var)

pour $0 \leq \text{Var} = \text{entier} \leq 256$ employer CHR\$ pour 1 caractère (Ln)
 $32768 \leq \text{Var} = \text{entier} \leq 32768$ employer MKI\$ pour 2 caractères (Ln)
Var = simple précision employer MKS\$ pour 4 caractères (Ln)
Var = double précision employer MKD\$ pour 8 caractères (Ln)

Reconversion chaînes vers numérique en relecture.

pour CHR\$	en retour	employer Var = ASC (Varn\$)
pour MKI\$	en retour	employer Var = CVI (Varn\$)
pour MKS\$	en retour	employer Var = CVS (Varn\$)
pour MKD\$	en retour	employer Var = CVD (Varn\$)

Exemple :

100 LSET A\$ = MKI\$ (A%)
110 LSET B\$ = MKS\$ (B!)
120 LSET C\$ = MKD\$ (C#)
130 LSET D\$ = CHR\$ (32)
140 LSET E\$ = MKI\$ (2022)

200 A% = CVI (A\$)
210 B! = CVS (B\$)
220 C# = CVD (C\$)
230 D = ASC (D\$)
240 E = CVI (E\$)

9.7 - LOF

$X = \text{LOF}(\text{bn})$

Cette fonction retourne dans la variable X le numéro du dernier enregistrement du fichier bn.

Exemple : IF LOF (1) $\geq$ 100 THEN 1000

9.8 - GET PUT (FIELD AD)

Pour plus de clarté, nous séparerons l'accès au fichier FIELD AD de ceux aux fichiers FI, FF, MU, MI, MF définis en préambule

PUT bn, rec
GET bn, rec

Ces instructions assurent l'écriture (PUT) ou la lecture (GET) de l'enregistrement n° rec du fichier dont le n° de Buffer est bn

Ces instructions doivent être bien sûr précédées d'un OPEN "R" ou OPEN"D", et d'un FIELD correspondant

L'instruction PUT doit être précédée du rangement dans le FIELD par LSET ou RSET

Exemple :
100 OPEN "R", 1, "FICHEX", 64
110 FIELD 1, 60 AS A\$, 4 AS B\$
120 LSET A\$ = "POUR L'EXEMPLE"
130 LSET B\$ = MKS\$ (12.48)
140 PUT 1, 1
150 GET 1, 1
160 G\$ = A\$: G = CVS (B\$)
170 PRINT G\$; G ; LOF (1)
180 CLOSE

Cet exemple fait le tour des commandes liées aux fichiers d'accès direct classique (FIELD AD)

Voici quelques pièges à éviter en les employant :

- oubli de LSET avant PUT. On ne retrouve donc pas ensuite à la lecture ce que l'on croit avoir écrit dans le fichier.

- Utilisation dans une instruction 'INPUT' d'une variable précédemment définie dans un 'FIELD'. Celle-ci se trouve alors désallouée du buffer correspondant.

- Une même variable a été utilisée dans plusieurs FIELD. Elle n'est affectée en réalité qu'au dernier FIELD exécuté.

- Chevauchement (sans le vouloir) des zones. (par des FIELD... multiples).

- On ne lit pas les informations suivant le type où elles ont été rangées. (par exemple une information écrite par MKI\$ doit être convertie à la lecture par CVI).

- Oubli de GET avant la modification d'une zone d'un enregistrement. La zone modifiée est bien retrouvée mais toutes les autres contiennent les valeurs de l'enregistrement présent dans le buffer au moment de la mise à jour.

9.9 - GET PUT (Fichiers MI, MU, FI, FF, MF)

GET bn, fp, IGEL (ou IGELSN)

PUT bn, fp, IGEL (ou IGELSN)

Les explications concernant les IGEL, IGELSN, RBA, REMRA et REMBA sont données Section 9-11

bn indique toujours le n° de Buffer défini dans l'OPEN pour ce fichier

fp indique la position initiale dans le fichier et peut prendre plusieurs formes

, , espace vide. si REMRA est validé et le fichier segmenté en enregistrements, le buffer pointe vers l'enregistrement suivant, sinon l'espace vide équivaut à *

* Le pointeur de fichier reste inchangé

Le pointeur de fichier est repositionné sur REMRA ce qui permet de repositionner le fichier sur l'enregistrement qui vient d'être traité. Une erreur est signalée si REMRA est invalidé.

\$ Le pointeur de fichier est repositionné sur REMBA ce qui permet de repointer vers la position dans le fichier de l'ancienne commande PUT ou GET avec fp = vide, *, #, \$, rn ou ! rba. Cette position REMBA peut se situer en milieu d'un enregistrement surtout en non segmenté, mais souvent REMBA = REMRA

% autorise la déclaration d'un PSEUDO FIELD (voir section 9.11)

& (avec PUT uniquement) force l'écriture du Buffer sur disque (celle-ci n'étant en effet pas systématiquement faite à chaque PUT, mais quand le buffer est plein, il peut être important, pour sauvegarder vite sur le disque, de forcer l'enregistrement immédiat de par cette option. Le pointeur de fichier n'est pas affecté par cette commande.

&& (avec PUT uniquement) force la mise à jour immédiate du DIRECTORY pour le fichier évitant ainsi un CLOSE et un autre OPEN, assurant ainsi la sécurité sans perturber l'état des variables du fichier. Bien sûr, un PUT bn, && force l'option &. Le pointeur de fichier n'est pas altéré

! rba rba est une valeur numérique comprise entre 0 et 16777215 et indique la position relative d'un caractère (octet) dans le fichier. L'opération GET ou PUT s'exécutera à partir de cette position.

Attention l'usage de cette fonction puissante peut aussi, si elle est mal employée, être désastreuse.

! % place le pointeur fichier sur l'EOF et transfère

! \$ rba positionne le pointeur sur le rba fixé dans la commande mais sans effectuer aucun transfert

! S idem mais pointeur sur l'EOF sans transfert

! # rba donne à l'EOF la nouvelle valeur rba.

rn indique le n° de l'enregistrement (0 - 32767). C'est le procédé connu des FIELD AD (random) et par extension, des fichiers FF ou MF ouverts avec m = I, R ou D. Car il faut bien sûr pour utiliser rn que le fichier soit segmenté en enregistrement.

Exemples : GET 1, ! \$, 1354
 GET 1, ! 1357, 1000
 PUT 1, &&
 GET 1, , 1000

Pour toute explication concernant les IGEL et IGELSN, voir section 9.11

9.10 - LOC

X = LOC (bn) FIELD, MF ou FF
 X = LOC (bn) \$ Tous types
 X = LOC (bn) % Tous types
 X = LOC (bn) ! Tous types
 X = LOC (bn) # Tous types

où **bn** représente toujours le n° de buffer attribué au fichier dans l'OPEN

LOC (bn) utilisable pour les fichiers FIELD, MF ou FF
 Cette fonction retourne dans X un entier (1 - 32767) désignant la position du pointeur de fichier, donc le n° d'enregistrement utilisé dans la dernière instruction PUT ou GET

Exemple : PUT 1,34
 X = LOC(1) donnera X = 34

LOC (bn) \$ cette fonction teste la fin de fichier (EOF). X prend la valeur - 1 si EOF est atteint ou dépassé, 0 sinon. LOC (bn) \$ diffère de la fonction EOF qui ne teste que EOF atteint (et non dépassé)

LOC (bn) % Cette fonction retourne dans X la valeur en octets relative (RBA) de l'EOF

Exemple : si un fichier contient 3263 caractères (octets)
 X = LOC (bn) % sera égal à 3263

LOC (bn) ! cette fonction retourne dans X la position courante

En fichiers segmentés (FIELD, MF, FF, MU) cette fonction retourne dans X un RBA = position du prochain enregistrement si REMRA validé
 = position courante si REMRA invalidé

En fichiers non segmentés, cette fonction retourne dans X le RBA de la position courante

LOC (bn) # Cette fonction retourne dans X le RBA égal au REMBA. Une erreur est déclenchée si REMRA est invalidé.

Exemple : si l'enregistrement complet ou partiel du fichier 1 a débuté à la position relative (RBA) 1667 et que le prochain enregistrement débute à la position relative 1707
 alors X = LOC (bn)! sera égal à 1707
 et Y = LOC (bn) # sera égal à 1667

Ces fonctions LOC (bn) utilisées avec les (PUT et GET) fp = rba et ! \$rba, permettent de tenir à jour des index d'accès fichiers pour accès direct même sur des fichiers non segmentés en enregistrement.

9.11.1 - L'IGEL

L'IGEL est une liste de caractéristiques de variables destinée à établir un lien entre les ordres d'écritures/lectures sur le disque et les variables à récupérer (un peu comme les données d'une instruction FIELD, le rangement en moins).

Chaque expression dans L'IGEL est séparée par une virgule et elle se termine par un point virgule. Elle peut s'enchaîner sur plusieurs lignes, mêmes séparées par des commentaires :

```
10 Exp 1 , Exp 2
20 REM "JE NE GENE PAS"
30 Exp 3 , Exp 4 ;
```

L'Expn de L'IGEL peut prendre 5 formes :

1 - Var : c'est la situation la plus courante, var obéit aux règles des variables BASIC ; cela peut être une donnée directe ou une variable de tableau. Il est fortement recommandé d'utiliser les attributs.

Exemple : A\$, AX ! (X , Y) , "3" , C%

2 - (len) var \$: len précèdent la variable de chaîne donne la longueur de cette variable, à l'écriture comme à la lecture.

len est facultatif en IGEL de fichier MF, MI, MU (séparateurs)

len est obligatoire en IGEL de fichier FF, FI

len peut être plus grand que la longueur de variable réelle :

en fichiers MF, MI, MU il indique la valeur maxi au-delà de laquelle on troncare mais en dessous c'est la variable seule qui génère la place.

en fichier FF, FI, c'est la place réelle réservée : en dessus on troncare, en dessous on remplit d'espaces.

3 - (len) \$ employé uniquement pour fichier FI, FF

len indique le nombre de caractères (ou octets) à sauter (sans les altérer pour un PUT). Pour un PUT sur un nouvel enregistrement (len) \$ = (len) #

4 - (len) # pour les fichiers FI, FF, et pour un GET, (len) # = (len) \$ et pour un PUT, génère une variable de longueur len remplie de $\emptyset$.

pour les fichiers MF, MI, MU, (len) # pour un GET, (len) # fait sauter d'une variable et pour un PUT, génère une variable de chaîne de longueur len remplie de $\emptyset$.

5 - Expn = , , un emplacement vide dans L'IGEL ne peut être utilisé que pour les fichiers MF, MI, MU et pour un GET uniquement, et réagit comme (len) # à savoir sauter une variable (ce qui est rendu possible simplement par les caractères de contrôle des fichiers du type "Mx").

Si une erreur se produit dans le traitement d'un IGEL, le message donnera la position de l'erreur dans L'IGEL.

9.11.2 - IGELSN

L'IGEL peut prendre place directement dans les commandes PUT et GET, mais il peut aussi, et c'est souvent préférable pour un emploi fréquent de le placer sur une (ou plusieurs) lignes séparées. Dans ce cas, les instructions PUT et GET doivent L'IGELSN c'est-à-dire le premier n° de ligne, point de départ de L'IGEL.

Exemple d'emploi d'IGEL et IGELSN

50 GET 1 , , A\$, B! , C #

100 A\$, C\$, DA% , (30)\$, CD%
110 "3" , X (I , J) ;

150 PUT 2 , I 58 , 100

9.11.3 - REMBA REMRA RBA

RBA Adresse en relatif par rapport au début du fichier de l'octet (ou caractère pointé)

Le 1638ième caractère d'un fichier possède un RBA de 1637 (car le 1er est compté 0)

Ø < rba < 16777215

REMRA mémorisation de l'adresse enregistrement

Utilisée dans les fonctions GET ou PUT avec fp = # ou loc (bn)\$, I , #

Le REMRA est l'adresse du 1er caractère de l'enregistrement courant (le dernier auquel on ait eu accès) exprimé en RBA (relatif octet)

REMBA mémorisation de l'adresse d'une variable

Utilisée dans les fonctions GET ou PUT avec FP = \$

Le REMBA est l'adresse du 1er caractère de la variable courante (la dernière auquel on ait eu accès) exprimé en RBA.

En cas de lecture ou écriture complète d'enregistrement
REMBA = REMRA

Le fait d'utiliser un fp = ! \$rba ou ! \$ % , ainsi que chaque OPEN , rend REMBA et REMRA invalidé.

le fait d'utiliser un fp = ! rba , ! % , rn , nul , pointe REMBA et REMRA à la position obtenue.

9.11.4 - LE PSEUDO FIELD

Les fichiers MF, MI, MU, FF et FI assurent donc le transfert des données via Un IGEL, soit directement cité dans l'instruction, soit sous une forme d'un IGELSN.

Si l'utilisateur désire utiliser la méthode de rangement par LSET et RSET, il a cette possibilité par l'intermédiaire du Pseudo FIELD avec $Fp = \%$

Exemple : **GET bn, %, IGEL (ou IGELSN)**
 GET 2, %, (30) A\$, 10 (C\$), D #

De par cette instruction (ou l'inverse par PUT) aucun transfert n'est effectué, simplement le buffer est préparé à recevoir les variables à la méthode des fichiers FIELD AD

- Les variables numériques sont inchangées
- Les espaces définis par (len)\$ et (len) # sont réservés
- Les variables de chaîne doivent être précédées de leur longueur réservée
- le transfert s'effectue par LSET et RSET pour les chaînes
- Cette instruction n'effectue aucun transfert sur disque et ne change pas la position des pointeurs fichiers.

9.12 - STRUCTURES D'ENREGISTREMENT COMPAREES DES FICHIERS MI, MU, MF, et FI, FF, P/I et FIELD AD

P/I	Var1	S	Var2	S	Var3	S	Var4	Varn
	S = Séparateurs : CR/LF/, / " par programmation enregistrement continu							
FIELD	Var1		Var2		Var3		Var4	
	Enregistrements de longueur définie (OPEN) Réservations par FIELD - CHARGEMENT PAR xSET - Con- version numérique							
MI	M	Var1	M	Var2	M	Var3	M	Var4
	Repérage sur enregistrement continu par marqueurs M Répartition par IGEL							
FI	Var1	Var2	Var3	Var4				
	Pas de repèresenregistrement continu Attribution d'espace et répartition par IGEL							
MF	M	Var1	M	Var2		M	Var3	M
	Segmentés en enregistrement de longueur définie (OPEN) Repérage par marqueurs M Répartition par IGEL							
FF	Var1	Var2	Var3	Var4	Var5			
	Segmentés en enregistrement de longueur définie (OPEN) Pas de repère Attribution d'espace et répartition par IGEL							
MU	SOR	M	Var1	M	Var2	SOR	M	Var3
	Segments en enregistrement de longueur variable Repérage par marqueurs M et SOR (début d'enregistrement) Répartition par IGEL							

EOF

10 - LE TRAITEMENT DES ERREURS

En cas d'erreur rencontrée dans le déroulement d'un programme, le BASIC retourne au mode direct, affiche l'erreur BASIC et le n° de ligne où elle s'est produite et éventuellement l'erreur DOS s'il en est question.

Il est souvent regrettable de se voir stopper en plein programme, alors que certaines erreurs sont prévisibles et donc peuvent être corrigées ou évitées (division par 0 par exemple)

Il est souvent nécessaire à la conception d'un programme pour un utilisateur non initié à l'informatique de pourvoir à la défaillance de l'opérateur en prévoyant tous les cas possibles d'erreur et leurs solutions éventuelles afin de ne pas stopper le déroulement du programme (surtout qu'en mode exécution forcée, le SYSTEME réinitialise automatiquement)

Les instructions suivantes permettent de solutionner le problème.

10.1 - ERROR ON ERROR GOTO

```
ERROR xx
ON ERROR GOTO aaaa
```

La première instruction permet d'afficher sur l'écran le message d'erreur xx ou en mode programme de brancher le déroulement au traitement d'erreur s'il y a lieu.

La seconde, utilisée en début de programme débute le traitement d'erreur : à partir de cette instruction, toute erreur rencontrée provoquera un saut à la ligne aaaa

ON ERROR GOTO 0 permet d'annuler le traitement d'erreur en cours

Exemple : 100 ON ERROR GOTO 1000

10.2 - ERL ERR

```
X = ERL
Y = ERR/2 + 1
```

L'instruction **ERL** retourne dans X le n° de ligne où s'est produit l'erreur.

L'instruction **ERR** retourne la valeur de l'erreur, mais pour avoir le code exact correspondant au message, il faut diviser par 2 et ajouter 1. Y contient donc le code d'erreur exact (erreur BASIC, pas DOS)

10.3 - RESUME

RESUME (aaaa)

Cette instruction termine une séquence de traitement d'erreur

RESUME sans n° de ligne, renvoie à ligne où s'est produit l'erreur

RESUME aaaa le programme se poursuit à la ligne aaaa

RESUME NEXT le programme se poursuit à la ligne suivant la ligne où s'est produit l'erreur

Exemple :

```
10  ON ERROR GOTO 100
20  INPUT "DIVISEUR ; X
30  PRINT 100/X
40  GOTO 20
100  IF (ERR/2 + 1) = 11 THEN "DIVISEUR = Ø
      INTERDIT"
      : RESUME 20
110  END
```

11 - L'ÉDITEUR PLEINE PAGE

Cet éditeur puissant et fort simple à utiliser, permet de supprimer de changer, d'insérer, de reproduire à l'intérieur de la partie de programme affichée sur l'écran.

L'entrée dans le mode ÉDITEUR se fait au travers de la commande EDIT

EDIT (aaaa) (—) (bbbb)
ou E (aaaa) (—) (bbbb)

aaaa ligne de départ (en haut de l'écran)
avec tiret jusqu'à la dernière ligne ou bbbb si précisé

bbbb ligne de fin (bas d'écran)
avec tiret devant depuis aaaa si précisé ou à défaut le début

E est équivalent à **EDIT**

Si l'écran est trop petit pour contenir l'ensemble des lignes demandées, le défilement ira toutefois jusqu'à bbbb inclus en bas d'écran et aucune ligne ne sera interrompue

Chacune des lignes affichées possède un caractère graphique en tête de ligne, utilisé par le mode ÉDITEUR pour repérer et valider les lignes modifiées, nous l'appellerons la **tête de ligne**. Cet indicateur effectue un pas à droite en cas de modification sur la ligne pointée.

Le déplacement du curseur est assuré sur l'ensemble de la page par les flèches    : essayez !

Pour quitter le mode ÉDITEUR, deux solutions : ENTER ou BREAK

BREAK pour quitter sans valider les modifications
Vous retournez en mode direct, le texte est intact

ENTER pour quitter en validant les modifications
Vous retrouvez le mode direct, mais le texte est corrigé

Dans ce dernier cas, les variables sont mises à zéro et les fichiers fermés. un CONT est devenu impossible.

11.1 - REMPLACER UN CARACTERE

Rien de plus simple : il suffit de positionner le curseur sur le caractère et de frapper le nouveau caractère au clavier. (L'écran est modifié. Rien n'est encore définitif avant "ENTER")

Toute une ligne de programme peut ainsi être recouverte par ce mode

11.2 - DETRUIRE UN CARACTERE

Le curseur doit être positionné sur le caractère à détruire. Appuyer alors sur SHIFT et  et l'ensemble de la ligne fera un pas à gauche, écrasant ainsi le caractère qui se trouvait sous le curseur. On peut ainsi éliminer des fins de ligne entières, bien que la commande CLEAR soit plus intéressante pour le faire

Exemple : ■ 10 PRINT "BONJOUR  MONSIEUR"
Curseur sur M et SHIFT  et on obtient
■ 10 PRINT "BONJOUR  MONSIEUR"
↑ notez le déplacement de l'indicateur

La fonction d'effacement est inefficace si :

- le curseur est sur une tête de ligne
- le curseur est après le dernier caractère de la ligne
- le curseur est en bas à droite de l'écran

11.3 - INSERER UN CARACTERE

Cette facilité permet d'insérer un ou plusieurs caractères sur une ligne ou en fin de ligne de programme

L'opération se déroule en 2 temps :

Il faut tout d'abord créer l'espace en plaçant le curseur derrière le caractère à ne pas modifier, puis presser SHIFT et . L'ensemble de la ligne à droite du curseur se déplace donc en créant un espace (ou plusieurs si je presse plusieurs fois ou longuement avec autorépétition)

Il faut ensuite dans cet espace ainsi aménagé, frapper ses propres caractères

Si besoin était, un interligne s'aménagera de lui-même poussant ainsi vers le bas les autres lignes

La fonction insertion est inefficace si le curseur est :

- sur une tête de ligne
- en fin d'écran
- après la 240ième position sur une ligne

Car une ligne ne peut contenir que 240 caractères

11.4 - EFFACEMENT FIN DE LIGNE

Un appui sur CLEAR efface tous les caractères à droite de la position du curseur

En positionnant le curseur juste après la tête de ligne, on peut effacer toute la ligne sur l'écran (mais pas en mémoire centrale) et en re-frapper une autre sans altérer la précédente.

En effet, les modifications s'effectuent à l'appui sur ENTER par lecture de l'écran. Si un n° de ligne a disparu, il ne sera donc pas altéré en RAM.

11.5 - REPRODUIRE UNE LIGNE COMPLETE

Rien de plus facile avec ce qui vient d'être expliqué

Il suffit de remplacer l'ancien numéro de ligne par le nouveau. L'ancienne ligne, absente de l'écran n'est pas altérée. La nouvelle est créée (tout cela si ENTER bien sûr).

Si l'on veut effacer la première, utilisez DELETE ou D.

11.6 - UN EXEMPLE commenté

EDIT 20 — 60

- 20 PRINT " BONJOUR MMONSIEUR"
- 30 PRINT " COMENT ALLER VOUS"
- 60 INPUT X\$

Amenez le curseur sous le 1er M de Monsieur et frappez SHIFT 

Amenez le curseur sur le M de COMENT et frappez SHIFT  la fin de ligne effectue un pas à droite ; frappez M maintenant

Amenez le curseur sur le R de ALLER et frappez Z puis —

- 20 PRINT " BONJOUR MONSIEUR "
- 30 PRINT " COMMENT ALLEZ-VOUS "
- 60 INPUT X\$

Maintenant ENTER validera vos modifications, BREAK non.

APPENDICE A : GLOSSAIRE TECHNIQUE DES MOTS EMPLOYES DANS CET OUVRAGE

(voir également le Manuel DOS)

- bit :** composant élémentaire de la donnée : état d'une ligne de donnée (parmi les 8 du Z80) peut prendre la valeur 0 ou 1.
- buffer :** espace de mémoire centrale servant de zone intermédiaire de stockage (tampon) entre les périphériques et le logiciel d'exploitation : Buffer clavier, buffer disque ...
- Caractère** (ou octet) : donnée élémentaire regroupant les états des 8 lignes du Z80 (bits), elle peut prendre 256 valeurs différentes (2^8), généralement exprimées en décimal (0 - 255) ou en hexa (00 M - FF H) (base 16).
- Carfich :** caractéristiques d'identification d'un fichier sur disque comprenant le nom, l'extension s'il y a lieu, le mot de passe s'ils sont validés, le n° du lecteur (facultatif) (voir manuel NEWDOS section 2, page 2.1).
- DOS :** Disk Operating Système - Système d'exploitation des disques : désigne l'ensemble du logiciel assurant l'interface entre l'électronique et les progiciels, les langages, et l'utilisateur. Le NEWDOS 80 . 2 . 0 est le DOS sous lequel travaille le SBASIC. Le C/PM3 est un autre DOS.
- DRIVE :** mot anglais pour désigner un lecteur de disquette. Couramment employé en micro-informatique DRIVE 0 = lecteur 0.
- EOF :** valeur de l'octet relatif (par rapport au 1er) marquant la fin d'enregistrement d'un fichier (End of file)
- EOL :** octet de valeur 0DH marquant la fin de la ligne de caractère (End of line).
- FCB :** groupe de 32 octets assurant dans le DIRECTORY d'une disquette, la gestion d'un fichier. Il en contient le nom, l'extension les mots de passe (accès et mise à jour), les éléments permettant d'identifier ses emplacements physiques (secteurs) sur la disquette. Ce bloc est transporté en mémoire centrale à l'ouverture d'un fichier pour que le DOS en assure la mise à jour (voir OPEN, CLOSE et &&) (File control block).
- HIMEM :** le DOS gère l'espace mémoire présent sur les cartes (d'origine 64 kilo octets = 64×1024 caractères). On peut lui imposer d'en laisser une partie hors de son autorité en haut de mémoire (pour la préserver d'être recouverte s'il elle contient les données importantes pour l'utilisateur). Cette frontière haute s'appelle le HIMEM. Une fonction DOS permet de la fixer. A défaut le HIMEM est au maximum disponible.
- IGEL :** voir section 9.11.1.
- IGELSN :** voir section 9.11.2.

- Listing :** ce mot, toujours d'origine anglaise, désigne l'édition de tout ou partie d'un texte, à l'écran ou sur imprimante.
- Octet :** voir Caractère.
- PORT :** le Z80, microprocesseur assurant la direction de l'électronique sur le GUEPARD, peut gérer en direct 16 lignes d'adresse et à l'aide de signaux de commandes particulières, il peut assurer des entrées et sorties sur 256 canaux appelés PORT au travers desquels peuvent circuler des données sans pouvoir y être stockées comme en RAM.
- Pointeur :** groupe de 2 octets désignant une adresse spécifique ayant une importance particulière dans le traitement en cours.
- RAM :** Random Accès Memory - Abréviation familière du langage informatique désignant la mémoire centrale, ou mémoire vive ou mémoire utilisable. En fait : Mémoire à lecture / écriture qui perd toute information si elle est hors alimentation.
- ROM :** Read only Memory - Abréviation familière du langage informatique désignant la mémoire dite morte, contenant souvent le langage moniteur et quelquefois un langage sur d'autres appareils. Son avantage est qu'elle est inaltérable.
- RBA,
REMRA
REMBA :** voir section 9.11.3.
- REGISTRE :** en micro-informatique, désigne une sorte de boîte aux lettres magnétique dont sont équipés la plupart des processeurs (micro, contrôleur de disques, d'écran, générateur de son, série ...) et dont l'état détermine le fonctionnement de ceux-ci. C'est par ce biais que l'on télécommande les actions de ces processeurs. Sur le microprocesseur lui-même, il désigne également des boîtes aux lettres à usage interne assurant la ventilation des données et adresses.
- SCROLLING** encore un mot anglais pour désigner le phénomène de remontée de tout l'écran d'une ligne lorsque le curseur atteint sa limite autorisée.

APPENDICE B

Commande	Section	Page	Commande	Section	Page
&H	7-12	7-2	DEFxxx	5-3	5-2
&O	7-13	7-2	DELETE	4-2	4-1
ABS	7-1	7-1	DI	4-3	4-2
AND	5-15	5-7	DIM	5-2	5-1
ASC	8-1	8-1	DU	4-3	4-2
ASC	9-6	9-9	EDIT	4-5	4-2
ATN	7-9	7-2	ELSE	5-7	5-3
AUTO	4-1	4-1	END	5-10	5-5
CAR	6-1	6-1	EOF	9-4	9-7
CHR\$	8-2	8-1	ERL	10-2	10-1
CHR\$	9-6	9-9	ERR	10-2	10-1
CLEAR	5-1	5-1	ERROR	10-1	10-1
CLM	6-12	6-6	EXP	7-4	7-1
CLOSE	9-2	9-5	FIELD	9-5	9-8
CLS	6-8	6-5	FIX	7-3	7-1
CLV	6-6	6-2	FLA	6-5	6-2
CMD "C"	4-18	4-8	FN	5-4	5-2
CMD "E"	4-19	4-8	FONCAR	6-3	6-1
CMD "F"	4-20	4-9	FONECR	6-2	6-1
CMD "J"	4-22	4-12	FOR	5-8	5-3
CMD "O"	4-21	4-11	FRE	5-18	5-8
CMD "R"	4-24	4-12	GET	9-8	9-10
CMD "S"	4-25	4-12	GET	9-9	9-11
CMD "T"	4-23	4-12	GOSUB	5-9	5-4
CMD	4-10	4-4	GOTO	5-9	5-4
CONSB	6-4	6-2	GRE	6-5	6-2
CONSH	6-4	6-2	IF	5-7	5-3
CONT	4-9	4-4	INKEY\$	5-12	5-5
COS	7-6	7-2	INP	5-14	5-7
CVD	9-6	9-9	INPUT #	9-3	9-6
CVI	9-6	9-9	INPUT	5-12	5-5
CVS	9-6	9-9	INSTR	8-10	8-2
DATA	5-5	5-2	INT	7-2	7-1
DEF FN	5-4	5-2	INV	6-5	6-2
DEFUSRn	5-13	5-6	KILL	4-13	4-5

Commande	Section	Page	Commande	Section	Page
LEFT\$	8-4	8-1	PUT	9-9	9-11
LEN	8-3	8-1	READ	5-5	5-2
LINE INPUT #	9-3	9-6	REF	4-15	4-7
LINE INPUT	5-12	5-5	REM	5-20	5-9
LIST	4-4	4-2	RENEW	4-16	4-8
LLIST	4-4	4-2	RENUM	4-14	4-5
LOAD	4-7	4-3	RESET	6-10	6-5
LOC	9-10	9-12	RESTORE	5-5	5-2
LOF	9-7	9-10	RESUME	10-3	10-2
LOG	7-10	7-2	RETURN	5-9	5-4
LPRINT	6-11	6-5	RIGHT\$	8-5	8-1
LSET	9-5	9-8	RND	5-6	5-3
MEM	5-17	5-8	RSET	9-5	9-8
MERGE	4-11	4-4	RUN	4-8	4-3
MID\$	8-6	8-2	SAVE	4-6	4-3
MKD\$	9-6	9-9	SET	6-10	6-5
MKI\$	9-6	9-9	SGN	7-11	7-2
MKS\$	9-6	9-9	SIN	7-7	7-2
NAME	4-12	4-5	SOUND	6-12	6-6
NEW	4-16	4-8	SQR	7-5	7-1
NEXT	5-8	5-3	STEP	5-8	5-3
NOR	5-15	5-7	STOP	5-10	5-5
ON	5-9	5-4	STR\$	8-8	8-2
OPEN	9-1	9-1	STRING\$	8-7	8-2
OR	5-15	5-7	TAB	6-7	6-3
OUT	5-14	5-7	TAN	7-8	7-2
PAUSE	5-11	5-5	THEN	5-7	5-3
PEEK	5-14	5-7	TIME\$	5-19	5-8
PLAY	6-12	6-6	TO	5-8	5-3
POINT	6-10	6-5	TROFF	4-17	4-8
POKE	5-14	5-7	TRON	4-17	4-8
POS	6-9	6-5	USING	6-7	6-3
PRINT #	9-3	9-6	USRn	5-13	5-6
PRINT	6-7	6-3	VAL	8-9	8-2
PUT	9-8	9-10	VARTPR	5-16	5-8

APPENDICE C : MESSAGES D'ERREURS COMMENTES

Rappel Code = ERR / 2 + 1

Code Message

1 NEXT SANS FOR

Un NEXT est rencontré sans que l'on puisse lui associer un FOR

Exemple : 10 A = 1
20 B = A + 2
30 NEXT
RUN
NEXT SANS FOR en 30

2 ERREUR DE SYNTAXE

Entrée de caractères de commande incorrecte : fautes d'orthographe, oubli de parenthèses, de points virgules, ...

Exemple : 10 PRINT "ABCD"
RUN
ERREUR DE SYNTAXE en 10 (guillemets finaux)

3 RETURN SANS GOSUB

Un RETURN est rencontré sans que l'on puisse lui associer un GOSUB.

Exemple : 10 PRINT "HOURRA"
20 FOR I = 0 TO 1000 : A = A + 1 : NEXT
30 RETURN
RUN
RETURN SANS GOSUB en 30

4 DATA EPUISEES

Un READ a épuisé les DATA, ou, n'en a pas trouvé du tout.

Exemple : 10 DATA 5, 3, 4
20 READ A, B, C, D
RUN
DATA EPUISEES en 20

5 FONCTION ILLEGALE

Ce message est signe d'un paramètre hors limite en numérique ou en traitement de chaîne
un indice négatif ou hors limite.
un operande négatif ou nul en LOG
un operande négatif en SQR
un USR sans DEFUSR
un mauvais operande en LEFT\$, RIGHT\$...

Exemple : 10 A = INP (330)
20 PRINT A
RUN
FONCTION ILLEGALE en 10

6 DEPASSEMENT

Nombre trop élevé (dans un dimensionnement de tableau par exemple).

```
Exemple : 10 DIM A (1111111)
          20 FOR I = 0 TO 1111111 : A (I) = 2 : NEXT
          RUN
          DEPASSEMENT en 10
```

7 DEBORDEMENT MEMOIRE

Plus assez de mémoire pour le programme ou les variables, ou trop de boucle FOR . NEXT en cours ou trop de GOSUB en cours.

8 LIGNE NON DEFINIE

Une commande fait appel à une ligne non définie (inexistante) : GOTO, GOSUB, DELETE, THEN-ELSE,...

```
Exemple : 10 READ A, B
          20 IF A = B GOTO 100
          30 DATA 1, 1
          RUN
          LIGNE NON DEFINIE en 20
```

9 INDICE HORS DIMENSION

Un élément de tableau est déclaré avec un indice ≥ 10 sans avoir été dimensionné ou avec un indice supérieur à celui du DIM.

```
Exemple : 10 DIM A (20)
          20 FOR I = 0 TO 30
          30 A (I) = A (I) + 1
          40 NEXT
          RUN
          INDICE HORS DIMENSION en 30
```

10 REDIMENSIONNEMENT DE TABLEAU

Deux ou plusieurs DIM font appel à la même variable indicé.

```
Exemple : 10 DIM A (300)
          20 DIM A (40)
          RUN
          REDIMENSIONNEMENT DE TABLEAU en 20
```

11 DIVISION PAR Ø

Une opération avec dénominateur à Ø est demandée et n'est bien sûr pas valable.

```
Exemple : 10 : A = Ø : B = 12
          20 : C = B/A
          30 : ? C
          RUN
          DIVISION PAR Ø en 20
```

12 ILLEGAL EN MODE DIRECT

Signale l'emploi de commande interdite en mode direct (comme INPUT, INPUT #, DEFFN, ...)

Exemple : INPUT "NOM"; C\$
 ILLEGAL EN MODE DIRECT

13 MELANGE DE TYPE DE DONNEES

Assignation d'une chaîne vers une variable numérique ou vice et versa.

Assignation d'une variable numérique dans une fonction à traitement de chaînes.

Exemple : 10 CLEAR 1000
 20 ? STRING \$ (10,20)
 30 ? "BONJOUR"
 RUN
 MELANGE DE TYPE DE DONNEES en 20

14 PLUS D'ESPACE POUR LES CHAINES

L'espace réservé (en haut de mémoire) pour les chaînes est insuffisant. Le CLEAR xxxx a été insuffisant.

Refaire un CMD "F = SASZ", yyyy. (voir section 4.20).

Exemple : 10 B\$ = STRING \$ (500, "A")
 20 C\$ = "MAMAN"
 30 END
 RUN
 PLUS D'ESPACE POUR LES CHAINES en 10

15 CHAINE TROP LONGUE

Assignation à une variable de chaîne d'une chaîne supérieure à 255 caractères.

Exemple : 10 CLEAR 1000
 20 A\$ = STRING \$ (250, "A")
 30 B\$ = STRING \$ (250, "B")
 40 C\$ = A\$ + B\$
 RUN
 CHAINE TROP LONGUE en 40

16 ENCHAINEMENT TROP COMPLEXE

Une série d'instructions imbriquées sur les chaînes se révèle trop complexe à traiter pour le BASIC. Il faut décomposer la formule en sous-traitement.

17 IMPOSSIBLE

Une demande de continuer (CONT) est demandée dans une situation impossible (modification de programme après un BREAK ou un STOP, après un END).

18 ERROR SANS RESUME

La fin du programme est atteinte sans avoir trouvé de RESUME sous traitement d'erreur.

Exemple : 10 ON ERROR GOTO 100
20 READ A
30 GOTO 20
40 DATA 1, 2, 3
100 ? "ERREUR" : RESTORE
RUN
ERREUR SANS RESUME en 100

19 RESUME SANS ERREUR

Une instruction RESUME est rencontrée dans le déroulement d'un programme sans avoir été aiguillée au préalable par un ON ERROR GOTO.

Exemple : 10 PRINT A
20 IF I = 4 THEN 10
30 RESUME
RUN
RESUME SANS ERREUR en 30

20 ERREUR NON REPERTORIEE

Un code d'erreur inexistant parvient au traitement d'erreur du BASIC.

Exemple : ERROR 41
ERREUR NON REPERTORIEE.

21 MANQUE D'OPERANDE (S)

Un ou plusieurs operands manquent dans une instruction.

Exemple : 10 A = 10
20 B = 20
30 C = A +
RUN
MANQUE D'OPERANDE (S) en 30

22 DONNEES FICHIER INCORRECTES

Les données en provenance d'un fichier ne sont pas correctes ou arrivent dans un ordre impropre.
La lecture d'un fichier sur disque ne se fait pas dans le bon ordre des variables ou suivant un ISEL non conforme à celui de la sauvegarde.

24 FONCTION USR INDEFINIE

Une fonction USRn est employée sans avoir été définie par un DEF USRn.

Exemple : 10.A = 10
20 X = USR Ø (A)
30 ? X
RUN
FONCTION USR INDEFINIE en 20

51 DEPASSEMENT DANS FIELD

L'ensemble des sous-segments déclarés dans un FIELD dépasse en longueur celle fixée globalement pour l'enregistrement dans l'OPEN (256 par défaut).

Exemple : 10 OPEN "R", 11, "TEST", 64
20 FIELD 1, 27 AS A\$, 62 AS B\$
RUN
DEPASSEMENT DANS FIELD en 20.

52 ERREUR INTERNE

Erreur interne, non répertoriée dans le BASIC DISQUE.

53 MAUVAIS NUMERO DE BUFFER

Utilisation dans l'OPEN d'un numéro de Buffer non réservé au chargement du BASIC.

Utilisation dans une instruction GET, PUT, PRINT #, INPUT #, d'un numéro de Buffer non ouvert par un OPEN.

Exemple : 100 OPEN "R", 2, "TEST"
110 FIELD 2, 60 AS A\$
120 GET 3, 1
RUN
MAUVAIS NUMERO DE BUFFER en 120.

54 FICHER NON TROUVE

Le fichier à charger ou à lire en séquentiel (OPEN OPTION "I") n'est pas trouvé sur le DRIVE s'il est cité ou sur les DRIVE en service.

55 MAUVAIS MODE D'ACCES FICHER

Une écriture (ou une lecture) d'un fichier sur disque est demandée en conflit avec le type de fichier déclaré dans l'OPEN correspondant

Exemple : 10 OPEN "I", 1, "TEST"
20 PRINT # 1, A\$
RUN
MAUVAIS MODE D'ACCES FICHER en 20

56 FICHER DEJA OUVERT

Une ouverture de fichier (OPEN) est effectuée avec un numéro de buffer déjà attribué

58 ERREUR DOS

Une erreur du SYSTEME n'est produite durant les accès disque. Le message d'erreur du DOS apparaît également en clair (se reporter au manuel DOS)

59 FICHER DEJA EXISTANT

Le fichier spécifié dans l'instruction est déjà celui en service.

62 DISQUETTE SATUREE

Il n'y a plus une seule granule de libre sur la disquette.

63 INPUT # APRES LA FIN DU FICHIER

L'EOF est atteint ou demande au fichier de restituer plus de variables qu'il n'en possède. Il faut surveiller l'EOF (voir section 9 - 3 et 9 - 4)

64 MAUVAIS NUMERO D'ENREGISTREMENT

Emploi d'un numéro d'enregistrement hors limite dans un PUT ou un GET ($\emptyset < n^{\circ} \text{ rec} < 32768$)

65 NOM DE FICHIER INCORRECT

Le "Carfich" employé n'est pas conforme. Surveillez votre syntaxe (voir manuel DOS page MN 2 - 1)

66 MELANGE DE MODE D'ACCES

67 PROGRAMME NON BASIC

Un LOAD, un MERGE ou un RUN est demandé pour un fichier qui n'est pas un programme BASIC.

68 PLUS D'ESPACE POUR LES FICHIERS

Toutes les entrées DIRECTORY sont prises (FDES) même s'il reste des granules, on ne peut plus ranger de fichiers sur la disquette (voir manuel DOS section 2 - 31 DDGA)

69 ECRITURE IMPOSSIBLE

Le message DOS affiche la raison de cette erreur, c'est-à-dire une disquette protégée contre l'écriture.

70 ACCES FICHIER REFUSE

Les mots de passe sont validés et ce n'est pas le bon, pour le niveau d'accès demandé.

71 DEBORDEMENT DE NUMERO DE SEQ.

72 DEBORDEMENT D'ENREGISTREMENT

La somme totale des réservations de place de variables déborde l'espace alloué à l'enregistrement dans l'OPEN

```
10 OPEN "R", 1, "TEST", "FF", 60
20 PUT 1, 1, 100
100 (80)B$, A !
RUN
```

DEBORDEMENT D'ENREGISTREMENT en 20

73 EXTENSION FICHIER IMPOSSIBLE

L'extension du fichier est impossible via une option ASE = N de la fonction CREATE (voir Manuel DOS section 2.13) Le numéro d'enregistrement dépasse celui prédéterminé

75 ERREUR PRECEDEMMENT AFFICHEE**76 LIGNE Ø INEXECUTABLE**

La fonction RENUM ne peut renuméroter une ligne Ø. Il faut la détruire ou la déplacer d'abord (DI. . . Section 4.3)

Exemple : Ø CLEAR 1000
2 DIM A (200)
4 PRINT "BONJOUR"
RENUM,
LIGNE Ø INEXECUTABLE

77 TYPE DE FICHIER INCORRECT

Une erreur de syntaxe s'est produite sur le type de fichier dans l'OPEN

Exemple : 10 OPEN " R " , 2 , " TEST " , " WF " , 80
RUN
TYPE DE FICHIER INCORRECT en 10

78 ERREUR DE SYNTAXE IGEL

Il s'agit en général d'un oubli de cloture de l'IGEL par un ; ou de séparer les variables par des virgules, ou une succession de lignes du même IGEL interrompue par une ligne autre que des commentaires.

Exemple : 90 GET 1 , * , 100 (après un OPEN)
100 (80) A\$, B # ,
110 PRINT A
120 BC ! , (20) D\$;
RUN
ERREUR DE SYNTAXE IGEL en 90 (1)

79 # VARIABLE ERREUR DE SYNTAXE IGEL

Une erreur de syntaxe s'est glissée dans la déclaration de la # ième variable de l'IGEL correspondant au PUT (GET)

Exemple : 90 PUT 1 , * , 100 (après un OPEN)
100 (80) B\$, CH — AMP , D #
RUN
2 VARIABLE ERREUR DE SYNTAXE IGEL en
90 (1)

80 # VARIABLE . len IGEL CHAINE ILLEGAL/MANQUANT

Dans l'IGEL intéressé, un len sur une variable de chaîne (voir SECTION 9-11) est manquant alors qu'il est obligatoire (FI, FF) ou illégal dans la #ième VARIABLE

Exemple : 90 PUT 1 , * , 100 (après un OPEN FI ou FF)
100 B # , A ! , D\$
RUN
3 VARIABLE . len IGEL CHAINE ILLEGAL/
MANQUANT en 90 (1)

82 LONGUEUR D'ENREGISTREMENT INCORRECTE

La longueur d'enregistrement spécifiée dans l'OPEN est
hors limite : $\emptyset$ ou > 256 en FIELD
 $\emptyset$ ou > 4095 en FF, MF, MU
 $\emptyset < L < 256$ en FIELD sans BASIC, xV
ou longueur absente alors qu'elle est obligatoire.

Exemple : 10 OPEN " R " , 2 , " TEST " , " FF "
RUN
LONGUEUR D'ENREGISTREMENT INCORRECTE
EN 10

83 STMT UTILISE DEUX NOMS DE FICHIER

84 MAUVAIS PARAMETRE DE POSITIONNEMENT FICHIER

Le paramètre **fp** du GET (ou du PUT) est mal employé, est illégal ou incorrect. revoir Section 9.9.

(1) Toute erreur détectée dans un IGELSN (ligne d'IGEL) renvoie, dans le message d'erreur, au numéro de ligne de l'appel (GET ou PUT)

APPENDICE D

LE CODE ASCII (sous NEWDOS 80 2.0 G)

Emploi de la Fonction BASIC CHR\$ (code)

Code	action
Ø	
1	
2	
3	
4	
5	
6	
7	
8	curseur gauche + effacement caractère (DEL)
9	
10	curseur bas (LF)
11	
12	
13	CR (ENTER) + LF
14	inx - pour "allumer" le curseur : POKE (&H 48B), valeur désirée.
15	inx - pour "éteindre" le curseur : POKE (&H 48B), 15
16	
17	
18	
19	
20	
21	
22	
23	
24	curseur gauche sans effacement
25	curseur droit sans effacement
26	curseur bas sans effacement.
27	curseur haut sans effacement.
28	retour début d'écran (HOME)
29	
30	effacement fin de ligne
31	effacement fin d'écran
32	espace

Autres codes, voir APPENDICE E : GENERATEUR DE CARACTERE.

APPENDICE E

LE GENERATEUR DE CARACTERE GUEPARD

Code	Car	Code	Car	Code	Car	Code	Car
0	■	32		64	à	96	—
1	<u>l</u>	33	l	65	A	97	a
2	<u>"</u>	34	"	66	B	98	b
3	<u>#</u>	35	#	67	C	99	c
4	<u>\$</u>	36	\$	68	D	100	d
5	<u>%</u>	37	%	69	E	101	e
6	<u>&</u>	38	&	70	F	102	f
7	<u>'</u>	39	'	71	G	103	g
8	<u>(</u>	40	(	72	H	104	h
9	<u>)</u>	41	)	73	I	105	i
10	<u>*</u>	42	*	74	J	106	j
11	<u>+</u>	43	+	75	K	107	k
12	<u>,</u>	44	,	76	L	108	l
13	<u>=</u>	45	=	77	M	109	m
14	<u>.</u>	46	.	78	N	110	n
15	<u>/</u>	47	/	79	O	111	o
16	<u>Ø</u>	48	Ø	80	P	112	p
17	<u>1</u>	49	1	81	Q	113	q
18	<u>2</u>	50	2	82	R	114	r
19	<u>3</u>	51	3	83	S	115	s
20	<u>4</u>	52	4	84	T	116	t
21	<u>5</u>	53	5	85	U	117	u
22	<u>6</u>	54	6	86	V	118	v
23	<u>7</u>	55	7	87	W	119	w
24	<u>8</u>	56	8	88	X	120	x
25	<u>9</u>	57	9	89	Y	121	y
26	<u>:</u>	58	:	90	Z	122	z
27	<u>;</u>	59	;	91	↑	123	{
28	<u><</u>	60	<	92	ç	124	ù
29	<u>=</u>	61	=	93	⌋	125	}
30	<u>></u>	62	>	94	^	126	ê
31	<u>?</u>	63	?	95	é	127	è

Code Ø à 31 non accessible par CHR\$(code) = Code de contrôle, Direct vidéo en langage machine.

Code	Car	Code	Car	Code	Car	Code	Car
128		160		192	<u>à</u>	224	<u>l</u>
129		161		193	<u>Á</u>	225	<u>a</u>
130		162		194	<u>B</u>	226	<u>b</u>
131		163		195	<u>C</u>	227	<u>c</u>
132		164		196	<u>D</u>	228	<u>d</u>
133		165		197	<u>E</u>	229	<u>e</u>
134		166		198	<u>F</u>	230	<u>f</u>
135		167		199	<u>G</u>	231	<u>g</u>
136		168		200	<u>H</u>	232	<u>h</u>
137		169		201	<u>I</u>	233	<u>i</u>
138		170		202	<u>J</u>	234	<u>j</u>
139		171		203	<u>K</u>	235	<u>k</u>
140		172		204	<u>L</u>	236	<u>l</u>
141		173		205	<u>M</u>	237	<u>m</u>
142		174		206	<u>N</u>	238	<u>n</u>
143		175		207	<u>O</u>	239	<u>o</u>
144		176		208	<u>P</u>	240	<u>p</u>
145		177		209	<u>Q</u>	241	<u>q</u>
146		178		210	<u>R</u>	242	<u>r</u>
147		179		211	<u>S</u>	243	<u>s</u>
148		180		212	<u>T</u>	244	<u>t</u>
149		181		213	<u>U</u>	245	<u>u</u>
150		182		214	<u>V</u>	246	<u>v</u>
151		183		215	<u>W</u>	247	<u>w</u>
152		184		216	<u>X</u>	248	<u>x</u>
153		185		217	<u>Y</u>	249	<u>y</u>
154		186		218	<u>Z</u>	250	<u>z</u>
155		187		219	<u>ı</u>	251	<u>-</u>
156		188		220	<u>ç</u>	252	<u>û</u>
157		189		221	<u>İ</u>	253	<u>-</u>
158		190		222	<u>ı</u>	254	<u>ê</u>
159		191		223	<u>é</u>	255	<u>è</u>

APPENDICE F : LES FICHIERS SBASIC voir section 9 - 12

Type de Fichiers	Enregistrements	Cde entrée/sortie	Séparateurs de données opérateur	Rangement des données	Reperage des enregistrements	Données numériques	Mise à jour possible
PRINT/INPUT (séquentiels)	non segmentés	PRINT # INPUT # LINE INPUT #	CR , LF " " utilisateur	même ordre à la lecture qu'à l'écriture		en ASCII	non
FIELD AD (accès direct)	segmentés $0 \leq L < 256$	GET , PUT	non	par FIELD LSET / RSET Conversions	logiciel par n°	en code conversion opérateur	oui $m = R$ ou D
MI	non segmentés	GET , PUT	oui logiciel	IGEL		en code conversion logiciel	non
FI	non segmentés	GET , PUT	non	IGEL		en code conversion logiciel	non
MF	segmentés $0 \leq L < 4095$	GET , PUT	oui logiciel	IGEL	logiciel	en code conversion logiciel	oui $m = R$ ou D
FF	segmentés $0 \leq L < 4095$	GET , PUT	non	IGEL	logiciel	en code conversion logiciel	oui $m = R$ ou D
MU	segmentés L = variable $0 \leq L < 4095$	GET , PUT	oui logiciel	IGEL	repères SOR	en code conversion logiciel	oui L = titrée $m = R$ ou D