LES LANGAGES DE PROGRAMMATION

# A la découverte du GS Basic

Emile Schwarz

UNE ÉDITION DE TREMPLIN MICRO

**EDITIONS JIBENA — TREMPLIN MICRO**
La Petite Motte, Senillé — 86100 CHÂTELLERAULT

**ISBN 2-901124-27-5**

# Un vrai Basic 65816 pour l'Apple IIGS !

*Le GS Basic ressemble à l'Applesoft, mais c'est autre chose. On peut parler de vrai Basic. C'est en tout cas le premier permettant effectivement une gestion mémoire étendue et des appels toolbox.*

*Vous allez vraiment le découvrir avec* ***Emile Schwarz*** *qui, pendant plusieurs mois, a défriché le terrain pour vous. Qui plus est, vous survolerez les instructions du GS Basic en français, ce qui n'exclut pas l'utilisation intensive de la documentation américaine (d'ailleurs indispensable).*

*TREMPLIN MICRO a déjà publié divers programmes en GS Basic... et poursuivra l'étude approfondie de cet excellent langage de programmation.*

*Et maintenant, à vous de tirer le meilleur parti de votre Apple IIGS !*

*Bonne programmation !*

***Guy-Hachette.***

Ce livre a été écrit en utilisant un Apple *IIGS* avec AppleWriter. Les fichiers ont ensuite été transférés sur un Macintosh Plus, puis imprimés sur une LaserWriter Plus à l'aide du matériel grâcieusement mis à ma disposition par **Sivea** Cannes.

Je tiens à remercier toute l'équipe de **Sivea** Cannes pour sa compréhension, sa gentillesse et leurs mini-cours de formation sur Macintosh.

**Emile Schwarz.**

# Introduction

Au sujet de..... 8

## A - Présentation.

Chaque page de cet ouvrage va décrire une instruction, étayée par un petit programme de démonstration.

On trouvera à chaque fois:
- le nom de l'instruction,
- le token et le type de l'instruction,
- une description de l'instruction,
- un renvoi à une ou plusieurs autres instructions,
- un ou plusieurs exemples.

## B - Syntaxe.

[ ] Les crochets signalent une option.

<> Le nom de chaque touche du clavier sera précédé de < et suivi de >.

Exemples:

```
CAT [/GSB]          LIST [,100]          AUTO [10] [,10]

<RETURN>              <ESC>                  <DELETE>
```

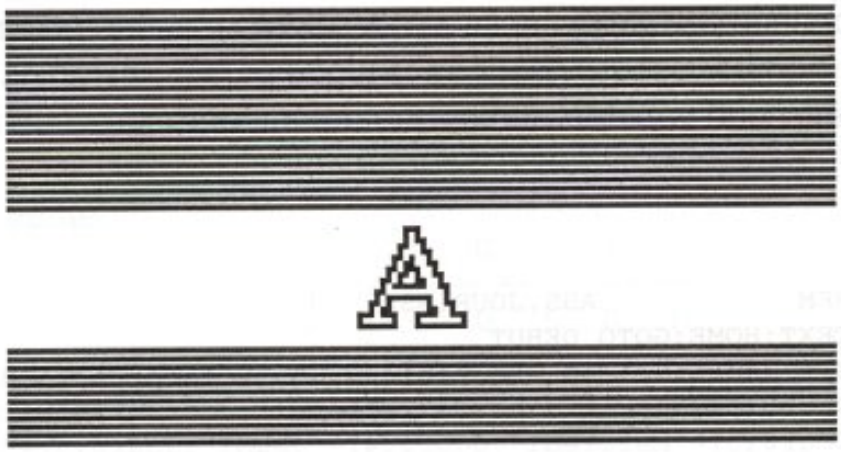
## C - En annexe on trouvera:

- la liste des erreurs **IIGS BASIC** avec leur numéro respectif,
- la liste des erreurs *ProDOS* avec leur numéro respectif,
- une table *ASCII* avec équivalence en hexadécimal, décimal, binaire.

**D - Tous les exemples** sont sur la disquette d'accompagnement classés alphabétiquement dans des sous-catalogues.

## E - Il est recommandé de consulter les ouvrages suivants:

1 - Le manuel de la boîte à outils (*Apple IIGS Toolbox reference* )
2 - *Apple numerics manual - Sane -*
3 - Le guide de l'utilisateur *Apple IIGS*.
4 - Le manuel de référence du **IIGS Basic**.



ABS() ..... 10
AND ..... 11
ANU() ..... 12
APPEND ..... 13
AS ..... 15
ASC() ..... 16
ASSIGN ..... 17
ATN() ..... 18
AUTO ..... 19
AUXIDà ..... 20

Token: DF 98 Fonction

Syntaxe: `a = ABS(n)`

La fonction `ABS()` donne la valeur absolue de son argument. La valeur absolue d'un nombre est sa valeur sans signe plus (+) ni moins (-). L'argument `n` peut être une variable numérique ou un nombre.

Exemple:

```
 10         REM              ABS.JOUR.GSB
 20         TEXT:HOME:GOTO DEBUT
 30         DIM Jour$(7):REM  Le tableau Jour$ comprendra 8 chaines
 40         Jour$(0)="-----":Jour$(1)="Lundi":Jour$(2)="Mardi"
 50         Jour$(3)="Mercredi":Jour$(4)="Jeudi":Jour$(5)="Vendredi"
 60         Jour$(6)="Samedi":Jour$(7)="Dimanche"
 70         A$="-----------------------------------------------"
 80         B$="!      A =      !   ABSolu  !    Nom du jour    !"
 90         RETURN
100 DEBUT: GOSUB 40
110         PRINT A$:PRINT B$:PRINT A$
120         FOR A%=-7 TO+7:REM On commence à -7 et on finit à +7
130             PRINT "!" SPC(6-(A%<0)) A% , "!" SPC(5) ABS(A%)SPC(5) "!";
140             PRINT SPC(5) "<" Jour$(ABS(A%)) ">" , "!"
150         NEXT
160 FIN:    END

RUN

-----------------------------------------------
!      A =      !   ABSolu  !    Nom du jour    !
-----------------------------------------------
!       -7      !     7     !      <Dimanche>   !
!       -6      !     6     !      <Samedi>     !
!       -5      !     5     !      <Vendredi>   !
!       -4      !     4     !      <Jeudi>      !
!       -3      !     3     !      <Mercredi>   !
!       -2      !     2     !      <Mardi>      !
!       -1      !     1     !      <Lundi>      !
!        0      !     0     !      <----->      !
!        1      !     1     !      <Lundi>      !
!        2      !     2     !      <Mardi>      !
!        3      !     3     !      <Mercredi>   !
!        4      !     4     !      <Jeudi>      !
!        5      !     5     !      <Vendredi>   !
!        6      !     6     !      <Samedi>     !
!        7      !     7     !      <Dimanche>   !
-----------------------------------------------
```

Token: DF 88 Opérateur logique

Syntaxe: `x = a AND b .`

L'opérateur logique AND est utilisé dans les suites IF ... THEN et ON ... GOTO ou ON ... GOSUB. On n'exécute l'instruction suivant le THEN, GOTO ou GOSUB que si les deux affirmations du AND sont exactes.

Table de vérité:

| a \ b | 0 | 1 |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 0 | 1 |

Exemple:

```
    5         REM      AND.GSB
   10 DEBUT:  TEXT:HOME:A=1:B=11:C=0:INVERSE
   20         PROC centre("Démonstration de l'instruction AND"):NORMAL
   30         VPOS=10:INVERSE
   40         PROC centre("IF A AND B THEN GOSUB vrai:ELSE:GOSUB faux")
   50         NORMAL:PRINT
   60 TEST1:  IF A AND B>10 THEN GOSUB vrai
   80             ELSE:GOSUB faux
   90         PRINT:INVERSE
  100         PROC centre("C = A = 12")
  110         NORMAL:PRINT
  120 TEST2:  C = A = 12
  130         IF C THEN GOSUB vrai
  140             ELSE:GOSUB faux
  150         LOCATE 22,0
  160 FIN:    END
  170 VRAI:   PROC centre("Le résultat de l'opération est vrai (1)")
  180         RETURN
  190 FAUX:   PROC centre("Le résultat de l'opération est faux (0)")
  200         RETURN
60500 CENTRE:DEF PROC centre(a$)
60510         LOCAL à%
60520         a% = 40-LEN(a$) DIV 2
60530         HPOS=a%:PRINT a$
60540         END PROC centre

RUN
Démonstration de l'opérateur logique AND

                    1 - IF A AND B > 10 THEN PRINT 'Chaine'
Les deux affirmations logiques sont vraies

                          2 - PRINT B = A = 12 AND C <> 0
B contient le résultat logique: B = 0
```

Token: DF C2 Fonction

Syntaxe: `A = ANU(taux,duree)`

La fonction annuité, `ANU(taux,duree)` calcule l'expression:

```
(1 - (1 + taux)(- duree)) / taux
```

Le calcul `ANU()` est plus exact que le calcul de l'expression en utilisant l'arithmétique normale et les opérations exponentielles.

La fonction `ANU()` est directement applicable au calcul de valeurs présentes et futures d'annuités normales.

Exemple:

```
5  REM       ANU.GSB
10 INPUT "Montant du prêt:.........................";A
20 INPUT "Nombre de remboursements:................";B
30 INPUT "Taux:....................................";C
40 PRINT "Montant de chaque 'Annuité':.............";A*ANU(B,C)
50 END

RUN

Montant du prêt:.........................1000
Taux:....................................10
Remboursements sur combien d'années:......10
Montant de chaque 'Annuité':.............100
```

Token: DF 84 Commande

Syntaxe:

```
1 - LIBRARY APPEND "*/TDFS/WINDOW.TDF"
2 - FOR APPEND "*/MONFICHIER"
3 - INVOKE APPEND "/GS/MONTRUC.SYS16"
```

**1 -** La commande APPEND associée à l'instruction LIBRARY permet d'ajouter un fichier .tdf dans le dictionnaire du **IIGS Basic**.

Exemple:

```
100 LIBRARY APPEND "*/TDFS/WINDOW.TDF"
110 LIBRARY APPEND "*/TDFS/MENU.TDF"
120 LIBRARY APPEND "*/TDFS/SANE.TDF"
```

**2 -** Si on désire écrire des données à la suite d'un fichier existant, on utilise FOR APPEND. Dans ce cas, l'écriture des données débute à la fin du fichier. Chaque accès successif a pour effet d'écrire là où le précédent s'est arrêté.

Exemple:

```
 10         REM      FOR.APPEND.GSB
 20         File$ = "/II.GSB/EXEMPLES.A/FOR.APPEND.TXT"
 30         CREATE File$, FILTYP=TXT
 40         OPEN File$, FOR APPEND, AS £10
 45 BOUCLE: BREAK ON
 50         FOR A% = 0 TO 9
 60             PRINT £10 "Ligne "A%
 70         NEXT
 80         CLOSE£10
 90         OPEN File$, FOR APPEND, AS £10
100         FOR A% = 10 TO 19
110             PRINT £10"Ligne "A%
120        NEXT
130        CLOSE£10
140        TYPE File$
150 FIN:   END
```

**ATTENTION:** FOR APPEND **NE FONCTIONNE PAS**. Il faut attendre la prochaine version qui, elle, donnera le résultat attendu.

**3 -** `INVOKE APPEND` est utilisé pour charger un ou plusieurs fichier(s) au format **OMF** (*Object Module Format*: c.a.d. des fichiers `.SYS16 - $B3` - créés à partir de MERLIN.16 ou d'APW) à la suite les uns des autres.

Si on tente de charger plusieurs fois le même fichier, chaque opération est prise en compte car `INVOKE APPEND` n'effectue aucun contrôle.

Voir les instructions `INVOKE`, `CREATE` et `OPEN`.

Token: DF 93

Syntaxe: `AS £n`

Le mot réservé `AS` est utilisé pour attribuer un numéro de référence à chaque fichier que vous ouvrez. Ce numéro servira à stocker, à charger ou encore à lire une information sur - ou à partir de - ce fichier.

La valeur de `n` est comprise entre `1` et `29`. `£0` est réservé pour `.CONSOLE`, et `£30` et `£31` pour le système.

Voir également la commande `OPEN`.

Exemple:

```
OPEN .PRINTER, AS £1 : OUTPUT £1 : LIST : OUTPUT £0 : CLOSE £1
```

- ouvre le fichier `.PRINTER` et lui attribue le numéro de référence `1`,
- envoie à l'imprimante le listing du programme **IIGSBasic** en mémoire,
- ferme le fichier référencé `1`.

```
OPEN .PRINTER, AS £1 : TYPE FICHIER.TXT, TO £1: CLOSE £1
```

- ouvre le fichier `.PRINTER`, lui attribue `1` comme numéro de référence,
- envoie le contenu du fichier texte `FICHIER.TXT` à l'impression,
- ferme le fichier imprimante référencé `1`.

Token: DF C8 Fonction

Syntaxe: `ASC("n")`

La fonction `ASC()` donne le code ***ASCII*** (1) décimal du premier caractère de la chaîne argument `n`.

Si la valeur de la chaîne est nulle, la valeur retournée sera: `-1`.

Voir `GET$`

Exemples:

```
 5         REM    ASC.GSB
10 DEBUT:  PRINT "Pressez une touche, Ctrl-C pour quitter  ";
20         GET$a$
30         a$=CHR$(ASC(a$)-128):PRINT
40         PRINT "Valeur ASCII de la touche pressée:       "ASC(a$)
50         IF ASC(a$)=3 THEN fin
60            ELSE:GOTO 10
70 FIN:    END

RUN
Pressez une touche, Ctrl-C pour quitter  A
Valeur ASCII de la touche pressée:      65
```

Remarque:

Contrairement à la logique **Apple**, `GET$` ramène un caractère *ASCII* négatif; ce qui nous oblige à opérer un recodage.

(1) *ASCII*: American Standard Code for Information Interchange.
En français: Code Standard Américain pour l'Echange d'Informations.
Se prononce 'aski'.

Token: D3

```
ASSIGN nomvolume,numport [,AUTO]
```

ASSIGN associe un nomvolume à un numéro de slot ou de port. **GS BASIC** définit six nomvolume. Chacun peut être changé en utilisant ASSIGN. Un nomvolume est un nom de fichier débutant par un point, suivi puis par une lettre, puis par un(e) ou plusieurs autres lettres ou chiffres.

On doit limiter la longueur du nomvolume à 15 caractères maximum (point compris) pour pouvoir définir 6 nomvolume supplémentaires. En effet, si on utilise des nomvolume trop long, on remplira rapidement le tampon mémoire où sont stockés les noms.

L'argument optionnel AUTO indique que le **GS BASIC** enverra après chaque *CR* (retour charriot) un *LF* (avancement de ligne) au périphérique. Lorsqu'un nomvolume a été défini par ASSIGN, le nouveau nom peut être utilisé par OPEN pour accéder au périphérique en entrée ou en sortie.

12 nomvolume peuvent être définis en utilisant ASSIGN (sont inclus les 6 noms par défaut). L'argument numport a une valeur qui correspond au numéro de slot du périphérique. Cette valeur sera comprise entre 0 et 7; 0 défini un périphérique nul, -1 annule le nomvolume de la liste.

Les six noms par défaut sont:

| Nom du périphérique | Slot | Auto-LF | Description |
|---|---|---|---|
| .CONSOLE | 3 | SANS | C3COUT1 |
| .PRINTER | 1 | AVEC | Imprimante |
| .MODEM | 2 | SANS | Modem |
| .MEMBUFR | - | SANS | pseudo périf. (buffer de 255 octets) |
| .NETPTR1 | 7 | AVEC | Appletalk |
| .NULL | 0 | SANS | lu = CR |

Exemple:

```
ASSIGN .PRINTER,-1:REM  Supprime .PRINTER.
ASSIGN .PRINTER,1:REM   Remet .PRINTER, mais sans AUTO-LF.
```

Token: DF AD

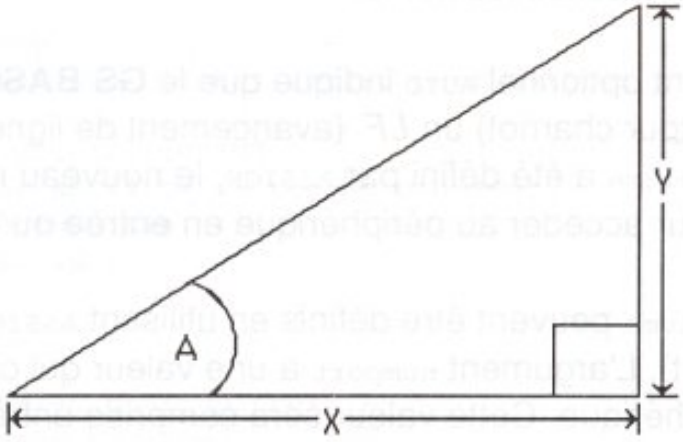
Fonction

Syntaxe: `PRINT ATN(n)`

La fonction `ATN()` calcule l'arctangente en radians du rapport `n`. Le résultat est une valeur comprise entre `-pi/2` et `+pi/2` radians (`PI = 3,141593`). `1` radian = `57,29577951308232087 5` degré

Pour convertir des radians en degrés, il faut multiplier la valeur exprimée en radians par `180` et diviser le résultat par `PI`.

Soit: `Degres = (radians * 180) / pi`




Dans la figure ci-dessus, l'angle `A` sera défini comme suit:

```
A = ATN(X/Y)
```

Exemple:

```
10 RADIANS = ATN(2)
20 DEGRES = (RADIANS * 180) / PI
30 PRINT "Angle en radians: "RADIANS,"degrés: "DEGRES
40 END
```

Cet exemple permet de trouver l'angle dont la tangente est égale à `2`: ce qui donne `1,107149 radians` ou `63,43495 degrés`.

Token: 80 Commande

```
AUTO [numligne] [,inc]
```

La commande AUTO génère automatiquement un numéro de ligne après chaque pression de la touche <RETURN>. La valeur par défaut du numéro de ligne et de l'incrément est 10. Un incrément supérieur à 1000 sera ignoré.

Avant d'entrer dans le mode AUTO, vous devez taper NEW; sinon vous serez en mode EDIT.

Pour sortir du mode AUTO, il suffira d'appuyer sur <ESC>

| | |
|---|---|
| `[numligne]` | est le numéro de la première ligne. |
| `[inc]` | est la valeur ajoutée à celle de la première ligne. |

Exemples:

| | |
|---|---|
| `AUTO` | début de la numérotation: 10, pas: 10 |
| `AUTO 100` | début de la numérotation: 100, pas: 10 |
| `AUTO 10,100` | début de la numérotation: 10, pas: 100 |
| `AUTO ,100` | début de la numérotation: 10, pas: 100 |

Token: DF EA Variable réservée

Syntaxe: `PRINT AUXIDà`

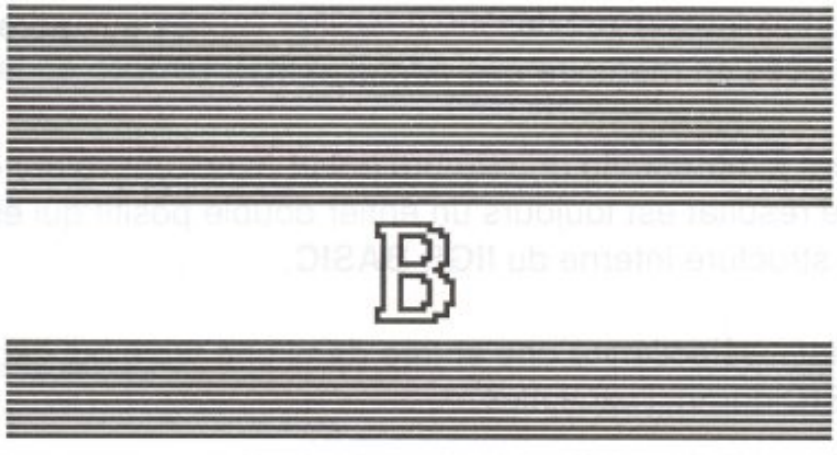
`AUXIDà` est une variable réservée, initialisée à chaque exécution d'un `OPEN`, `LOAD`, `SAVE`, `RUN`, `CHAIN`, `COPY` ou de la fonction `FILE?`.

`AUXIDà` retourne la valeur du champ sous-type du catalogue du dernier fichier référencé par une de ces commandes.

La valeur retournée sera du type entier double non-signé.

Exemple:

```
LOAD /GSB/DEMOS/PICS:PRINT AUXIDà
32767


10        REM   AUXID.GSB
20 DEBUT: PRINT "Démonstration de la variable AUXIDà"
30        PRINT "AUXIDà = "AUXIDà
40 FIN:   END

RUN
Démonstration de la variable AUXIDà
AUXIDà = 0
```



BASICà()..... 22
BDF..... 23
BREAK..... 24
BTN()..... 25

Token: DF AE

Fonction

Syntaxe: `PRINT BASICà(n)`

La fonction `BASCICà()` est destiné à faciliter l'accès aux adresses de certaines ressources en mémoire que donne le **IIGS BASIC**.

L'argument `n` représente un nombre entier avec une valeur comprise entre `0` et `255`; le résultat est toujours un entier double positif qui est l'adresse d'une structure interne du **IIGS BASIC**.

L'argument `n` sélectionne une entrée dans une table qui est conservée par le **IIGS BASIC** durant l'exécution.

Exemple:

```
10 REM      HELLO1.GSB
20 PRINT "Apple IIGS BASIC V1.0 BETA 4.0 10 Sept 1987"
30 PRINT "Auteur: "; VAR$ ( BASICà (0))
40 PRINT "Contributions: "; VAR$ ( BASICà(1))
50 PRINT "Contribution majeure:  "; VAR$ ( BASICà(2))
60 END

RUN

Apple IIGS BASIC V1.0 BETA 4.0 10 Sept 1987
Auteur: John O. Arkleya
Contribution: David Eyes, Taylor Pohlman, Dan Wendin, Clayton Lewis, Bill
Goldberg, Jim Merritt, Tom Chavez, Doug Thom
Contribution majeure: Dan Dennman, Charles S. Mauro, Joa, Randy Wigginton
```

Token: DF 91 Basic Data File Code mnémonique

Code mnémonique des fichiers variables du **IIGS Basic**. Il a pour valeur 173.

Utiliserer ce code mnémonique ou sa valeur DECIMALE avec `CREATE ... FILTYP=`

Voir en annexe la liste des types de fichiers.

Exemples:

```
CREATE  /NOMVOLUME/FICHIER, FILTYP=173
CREATE  /NOMVOLUME/FICHIER, FILTYP=BDF
```

Token: 86

Syntaxe:

```
BREAK ON
BREAK OFF
```

`BREAK OFF`, inséré dans un programme, inhibe l'effet du `Control-C` tapé sur le clavier.

Pour rétablir la prise en compte d'un `Control-C`, il suffit de mettre un `BREAK ON`, qui est le mode par défaut.

Exemple:

```
10          REM       BREAK.GSB
20 DEBUT:   BREAK OFF : REM On interdit le CTRL-C
30          PRINT "Interdiction du CTRL-C"
40          FOR A = 0 TO 250 : PRINT ".";:NEXT
50            PRINT : PRINT "Et maintenant: BREAK ON!"
60 BOUCLE:    BREAK ON : REM  On rétablit le CTRL-C
70            FOR A = 0 TO 250 : PRINT ".";:NEXT
80               PRINT : PRINT "C'est terminé..."
90 FIN           END

RUN
Interdiction du CTRL-C
.....................................................................
.....................................................................
.....................................................................
...........
Et maintenant: BREAK ON!
.....................................................................
.....................................................................
.............

PROGRAM INTERRUPTED
```

Token: DF 9D

Syntaxe: `PRINT BTN(0)`

`BTN()` retourne l'état de trois entrées, celui-ci étant renvoyé sous forme de `0` ou `1`. Les divers périphériques utilisant l'état de ces entrées comprennent les poignées de jeux, touches `Option` et `Pomme`.

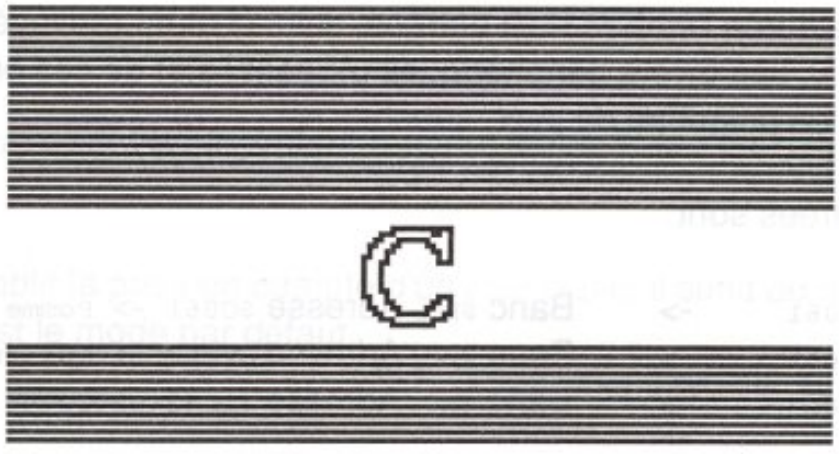
Ces entrées sont:

`BTN(0) - $E0C061` -> Banc `$E0`, Adresse `$C061` -> `Pomme`
`BTN(1) - $E0C062` -> Banc `$E0`, Adresse `$C062` -> `Option`
`BTN(2) - $E0C063` -> Banc `$E0`, Adresse `$C063` -> `Joystick`

Exemple:

```
 10 REM           BTN.GSB
 20 Actif$="activée,   valeur: "
 30 Inactif$="inactivée, valeur: "
 40 Pomme$="BTN(0) - Touche Pomme:  "
 50 Option$="BTN(1) - Touche Option: "
 60 Joystick$="BTN(2) - Joystick:      "
 70 PRINT "Appuyez sur la touche Pomme ou Option"
 80 PRINT "   pour modifier l'état de BTN(n)":PRINT
 90 FOR A=0 TO 100:NEXT:REM Boucle d'attente!
100 A=BTN(0):REM Touche Pomme
110 B=BTN(1):REM Touche Option
120 C=BTN(2):REM Joystick ...
130 IF A THEN PRINT Pomme$;Actif$;A : ELSE PRINT Pomme$;Inactif$;A
140 IF B THEN PRINT Option$;Actif$;B : ELSE PRINT Option$;Inactif$;B
150 IF C THEN PRINT Joystick$;Actif$;C : ELSE PRINT Joystick$;Inactif$;C
110   END

RUN

Appuyez sur la touche Option ou Pomme
   pour modifier l'état de BTN(n)

Touche Pomme:   inactivée, valeur 0
Touche Option:  inactivée, valeur 0
Joystick:       activée,   valeur 1
```

Le programme renvoie la valeur `0` ou `1` selon que l'on a pressé une ou plusieurs touches. Dans cet exemple, on a appuyé sur aucune touche, et il n'y a pas de Joystick connecté.




CALL et _ ..... 27
CALL% ..... 28
CAT ..... 29
CATALOG ..... 30
CHAIN ..... 32
CHR$() ..... 33
CLEAR ..... 34
CLOSE ..... 35
COMPI() ..... 36
CONT ..... 37
CONV() ..... 38
COPY ..... 40
COS() ..... 41
CREATE ..... 42

Token: B5 CALL
B6 _

Fonction

Syntaxe:

```
CALL nomlibrairie [parametre1] [,parametre2]
_    nomlibrairie [parametre1] [,parametre2]
```

CALL exécute la procédure nommée de la boîte à outils de l'*Apple IIGS*. Les outils ou leur définition d'interface sont chargés par LIBRARY et doivent être initialisés par votre programme **IIGS BASIC**.

Le numéro de fonction, le numéro d'outil, les paramètres, la taille et le type, - s'il y en a - sont pris dans la librairie par recherche du nomlibrairie dans le dictionnaire. Les définitions de chaque outil sont chargées par l'instruction LIBRARY à partir d'un fichier TDF (FILTYP= TDF). TDF = *Toolbox Definition File* (fichier de définition de la boîte à outils).

Le mot réservé CALL peut être remplacé par le caractère _ (souligné). Les TDF pour les outils doivent être chargé dans le dictionnaire avec l'instruction LIBRARY avant d'exécuter un CALL nomlibrairie; sinon, on obtient le message suivant:

```
?UNDEF'D PROC/FUNCTION ERROR
```

Le *Tool Locator* est utilisé pour appeler les différentes fonctions d'outils. On trouve dans le sous-catalogue /GSB/TDFS/ les fichiers '.tdf' pour presque tous les outils de votre disquette système. Les nomslibrairies ressemblent autant que possible, à ceux utilisés dans *The Apple IIGS Toolbox Reference Manual*.

Exemple:

```
_WindStartUp(myid%) : REM Démarre le Window Manager avec n° identificateur

_RefreshDeskTop(0) : REM Redessine l'écran en noir

_MoveTo(15,15):REMPlace les coordonnées en 15,15 avant l'écriture d'un message
_DrawString(VARPTR(msg1!(0))):REM Ecrit le message.
```

Voir également les instructions LIBRARY, FILTYP= et le *Apple IIGS Toolbox Reference Manual*.

Token: B4

Syntaxe:

```
CALL% numfonctionoutil,numoutil,tailleresult([parm1] [,parm2])
```

`CALL%` exécute la fonction `numfonction` de l'outil `numoutil` que l'on a précédemment initialisé. Le numéro de fonction doit être donné en premier, suivi du numéro de l'outil et, éventuellement de la taille en mots de l'argument renvoyé par l'outil sur la pile. Cet argument est pris de la pile du microprocesseur et les 16 premiers mots (32 octets) sont stockés dans la *pile buffer* et sont accessibles via la fonction `R.STACK`.

Les `numfonction`, `numoutil` et `tailleresult` doivent avoir une valeur comprise entre 0 et 255 ou une erreur `?ILLEGAL QUANTITY ERROR` apparaîtra. Le *Tool Locator* est utilisé pour dispatcher les appels aux fonctions. Ayez toujours le manuel de référence de la boîte à outils sous les yeux avant chaque appel à une fonction de la *toolbox* .

Exemple:

```
CALL% 84,4,1(RECT%(0))
```

Token: 96 Commande

Syntaxe:

| | |
|---|---|
| `CAT [.D1]` | De `.D1` à `.Dn n` = nombre de lecteurs de disquettes. |
| `CAT [/GSB/]` | Donne le CATalogue du volume. |
| `CAT [/GSB/TDFS]` | Donne le CATalogue du sous-catalogue. |
| `CAT [*]` | Donne le CATalogue du disque de démarrage. |
| `CAT [0]` | Donne le CATalogue du volume actuel. |
| `CAT [n]` | Donne le CATalogue du `PREFIX n`. |

`CAT` affiche à l'écran sur 40 colonnes: le nom, le type, la grandeur en kilo-octets, la date de dernière modification de chaque fichier contenu dans la disquette actuelle ou du préfixe actif. Voir également `CATALOG` et `DIR`. On peut aussi consulter la commande `PREFIX`.

Exemples:

1 - En mode direct:

```
CAT /GSB

/GSB/
NAME            TYPE  KBYTES  MODIFIED

*PRODOS          SYS    19.5  14-Jul-87
*SYSTEM          DIR     0.5  27-Aug-87
*GSBASIC.SYS16   S16    96.5  11-Sep-87
 FINDER.ROOT     $C9     0.5  17-Oct-87
 FINDER.DEF      $C9     0.5  18-Oct-87
*TDFS            DIR     1.5  11-Sep-87
*ICONS           DIR     0.5  11-Sep-87
*DEMOS           DIR     0.5  13-Oct-87
*PICS            GSB     7.5  10-Sep-87
*GSB.HELLO       GSB     0.5  10-Sep-87
 FINDER.DATA     $C9     0.5  11-Sep-87

 BLOCKS FREE: 186   BLOCKS USED: 1414
```

2 - En mode différé:

```
10 CAT "/APPLE.WRITER/"
```

Token: A7 Commande

Syntaxe:

| | |
|---|---|
| `CATALOG [.D1]` | Donne le CATALOGue du volume `.D1` |
| `CATALOG [/GSB/]` | Donne le CATALOGue du volume. |
| `CATALOG [SYSTEM]` | Donne le CATALOGue du sous-catalogue. |
| `CATALOG [/GSB/SYSTEM]` | Donne le CATALOGue du sous-catalogue. |
| `CATALOG [*]` | Donne le CATALOGue du disque de démarrage. |
| `CATALOG [0]` | Donne le CATALOGue du volume actuel. |
| `CATALOG [n]` | Donne le CATALOGue du `PREFIX n`. |

`CATALOG` affiche à l'écran sur `80` colonnes: le nom, le type, la grandeur en kilo-octets, la date et l'heure de dernière modification, la date et l'heure de création, la longueur en octets et le sous-type de chaque fichier contenu dans la disquette actuelle ou du préfixe actif. Voir également `CAT` et `DIR`.

Si la commande `CATALOG` est envoyée seule, le contenu de `PREFIX$` sera utilisé comme préfixe courant.

Voir les commandes `OPEN`, `FILTYP=`, `AS`, `OUTPUT`, `CLOSE` et `OUTREC`.

Exemples:

1 - En mode immédiat:

```
CATALOG *

/GSB/
NAME            TYPE  KBYTES  MODIFIED          CREATED         ENDFILE SUBTYPE

*PRODOS          SYS     19.5  14-Jul-87 13:50  14-Jul-87 13:50   19200
*SYSTEM          DIR      0.5  27-Aug-87 12:49  23-Jul-87 09:45     512
*GSBASIC.SYS16   S16     96.5  11-Sep-87 15:35  11-Sep-87 15:32   98304
 FINDER.ROOT     $C9      0.5  17-Oct-87 11:48  11-Sep-87 15:59       8
 FINDER.DEF      $C9      0.5  18-Oct-87 14:01  18-Oct-87 13:03      82
*TDFS            DIR      1.5  11-Sep-87 11:53  11-Sep-87 11:47    1536
*ICONS           DIR      0.5  11-Sep-87 11:59  23-Jul-87 09:57     512
*DEMOS           DIR      0.5  13-Oct-87 21:50  11-Sep-87 11:54     512
*PICS            GSB      7.5  10-Sep-87 19:27  17-Jun-87 09:22    6921
*GSB.HELLO       GSB      0.5  10-Sep-87 20:42  10-Sep-87 16:13     320
 FINDER.DATA     $C9      0.5  11-Sep-87 17:26  11-Sep-87 15:59     260

 BLOCKS FREE: 186   BLOCKS USED: 1414   TOTAL BLOCKS: 1600
```

2 - En mode différé:

```
200 CATALOG "/SYSTEM.DISK/SYSTEM"
```

3 - Dans un fichier:

```
5 REM       CATALOG.GSB
10 OPEN "/II.GSB/C/GS.PAINT", FILTYP=TXT AS £1
20 OUTPUT £1 : CATALOG "/GS.PAINT/"
30 OUTPUT £0 : CLOSE £1
40 TYPE "/II.GSB/C/GS.PAINT"
50 END
```

Le catalogue de `/GS.PAINT/` sera stocké dans le fichier `GS.PAINT` (du type `TXT` avec `1` comme n° d'identification) dont le contenu est affiché à l'écran.

4 - A l'imprimante:

```
OPEN .PRINTER, AS £1
OUTPUT £1 : CATALOG /APW/
OUTPUT £0 : CLOSE £1
```

Remarque: Il ne faut **JAMAIS** utiliser la commande `CLOSE £0` sous peine de devoir recharger le **IIGS Basic**.

Token: A6

Syntaxe:

```
CHAIN nomfichier [,numligne]
CHAIN nomfichier [,label]
```

L'instruction CHAIN charge et exécute un programme tout en conservant les variables déclarées dans le programme en cours sans fermer quelque fichier précédemment ouvert. Le programme nomfichier s'exécute à partir de l'option ,numligne ou ,label. Si le programme chaîné utilise DIM avec un tableau déjà défini dans le programme appelant, le message:

```
?DUPLICATE DEFINITION ERROR
```

s'affichera.

Exemple:

```
 5 REM       CHAIN.GSB
10 PRINT "Ceci est le programme CHAIN.GSB"
20 A$ = "Essai 1"
30 PRINT A$
40 CHAIN "CHAIN1.GSB"
50 END

 5 REM       CHAIN1.GSB
10 PRINT "Ceci est le programme CHAIN1.GSB"
20 A$ = "Essai 2"
30 PRINT A$
40 END

RUN

Ceci est le programme CHAIN.GSB
Essai 1
Ceci est le programme CHAIN1.GSB
Essai 1        Essai 2
```

On constate que l'on passe du programme CHAIN.GSB à CHAIN1.GSB sans perdre le contenu de la variable A$. On peut ensuite recharger CHAIN.GSB, éditer la ligne 40 et la changer en: CHAIN "CHAIN1.GSB,20" <RETURN>, puis faire RUN et voir la différence!

Token: DF C0 Fonction

Syntaxe: `PRINT CHR$(n)`

La fonction `CHR$()` renvoie le caractère *ASCII* correspondant à la valeur de son argument `n`, comprise entre `0` et `255`. Voyez en annexe la table des codes *ASCII*.

**Attention**: certains codes ont pour effet de bloquer le système; citons pour exemple `PRINT CHR$(21)` qui fait passer l'écran en 40 colonnes mais oblige à recharger le **IIGS Basic**.

1 - Mode immédiat:

```
PRINT CHR$(77)
Z
```

2 - Mode différé:

```
5 REM       CHR.GSB
10 FOR A = 32 TO 126: REM          Envoie les caractères
20 PRINT CHR$(A);: REM             commence par espace <ESPACE>
30 NEXT : REM                      et termine par tréma <¨>

RUN

 !"£$%&'()*+,-./0123456789:;<=>?àABCDEFGHIJKLMNOPQRSTUVWXYZ°
ç§^ _`abcdefghijklmnopqrstuvwxyzéùè¨
```

Message d'erreur possible:

```
?ILLEGAL QUANTITY ERROR
```

Token: CF

Commande

Syntaxe:

```
CLEAR [nnnnnn]
CLEAR [LIBRARY]
CLEAR [INVOKE]
```

La commande CLEAR efface les pointeurs de variables et de tableaux. Elle permet donc de passer autant de fois que vous le désirez dans une ligne contenant: DIM A$(100)

Elle permet également de spécifier à votre ordinateur la place en octets que vous voulez réserver pour les chaînes. L'argument optionnel nnnnnn a une valeur (x * 256) +1. Il précise la taille du segment mémoire à réserver. Pour connaître la place disponible, utilisez la commande FRE.

Vous pouvez utiliser CLEAR INVOKE et CLEAR LIBRARY pour effacer les segments de mémoire attribués précédemment et en rendre le contrôle au *Memory Manager*.

Exemple 1:

```
 10 REM      CLEAR.GSB
 20 B$="° passage":HOME:LOCATE 1,17:PRINT "Démonstration de l'instruction
    'CLEAR'"
 30 DIM A$(5):B=12:C=13:POKE 768,D+1
 40 C$="=============================================================="
 50 PRINT C$
 60 PRINT "* Valeur A: * Valeur B:  * Valeur C:    *          Nombre    *"
 70 PRINT "*   1 à 6   *   12       *      13       *       de passages *"
 80 PRINT C$
 90 A$(0)="1":A$(1)="2"
100 A$(2)="3":A$(3)="4"
110 A$(4)="5":A$(5)="6"
120 FOR A% = 0 TO 5
130   PRINT "*      "A$(A),B,C,
140   PRINT PEEK(768);B$,"  *"
150 NEXT
160 PRINT C$:PRINT
170 INPUT "Voulez-vous recommencer? (O/N) ";X$:PRINT
190 IF X$="O" THEN CLEAR:B$="° passage":D=PEEK(768):GOTO 30
200 END
```

Exemple 2:

```
10 CLEAR 65025
20 PRINT "Mémoire allouée par l'utilisateur: ";FRE;" octets"
30 END
```

Token: AE

Commande

Syntaxe:

```
CLOSE   [£numfichier1] [,£numfichier2]
```

L'instruction CLOSE ferme tous les fichiers ouverts. L'option £n ne fermera que le fichier correspondant à ce numéro. Avant de terminer un programme par END, on doit fermer tous les fichiers ouverts.

On peut fermer plusieurs fichiers en utilisant l'option £n, £n1, £n2 etc ...

Tout fichier ouvert sera fermé si on utilise les commandes suivantes:

```
LOAD, CLEAR, NEW, RUN.
```

CHAIN ne ferme aucun fichier.

Exemples:

```
OPEN .PRINTER, AS £1 : OUTPUT £1 : LIST : OUTPUT £0 : CLOSE

OPEN .PRINTER, AS £1 : OUTPUT £1 : LIST : OUTPUT £0 : CLOSE £1
```

Dans le premier cas, on ferme tous les fichiers; dans le second, on ferme seulement le fichier référencé 1 (.PRINTER dans ce cas).

Token: DF C3

Syntaxe: `COMPI(taux,duree)`

La fonction `COMPI()` est directement applicable au calcul de valeurs présentes et futures d'intérêts composés.

Le calcul `COMPI()` est plus exact que le calcul de l'expression en utilisant l'arithmétique normale et les opérations exponentielles.

Voyez la fonction `ANU()`.

Token: DB Commande

Syntaxe: CONT

La commande CONT (continue) relance l'exécution du programme là où un STOP ou un CTRL-C envoyé du clavier l'a stoppée. Contrairement à RUN qui met à zéro toutes les variables et démarre l'exécution à partir de la première ligne du programme, CONT n'affecte pas le contenu des variables et reprend l'exécution à la ligne qui suit l'interruption.

Exemple:

```
 5 REM   CONT.GSB
10 PRINT "Ceci est un test de la commande CONT"
15 PRINT "Tapez CONT pour continuer"
20 STOP
30 PRINT "Le test s'est déroulé suivant le plan prévu."
40 END

RUN

Ceci est un test de la commande CONT
Tapez CONT pour continuer

PROGRAM INTERRUPTED IN 20

CONT
Le test s'est déroulé suivant le plan prévu.
```

On peut également essayer cette instruction avec le programme CONV.GSB présent sur la disquette.

Token:

**1** - CONV( DF CD
**2** - CONV$ DF CF
**3** - CONV£( DF CB
**4** - CONV&( DF CE
**5** - CONVà( DF CC
**6** - CONV%( DF D0

Syntaxe: `a = CONV(n)`

**1** - CONV évalue l'argument n et retourne une valeur réelle en simple précision. La valeur retournée peut être assignée à une variable entière.

**2** - CONV$ est utilisé avec une chaîne; l'effet est le même qu'avec la fonction VAL.

**3** - CONV£ évalue l'argument et retourne une valeur en double précision.

**4** - CONV& évalue l'argument et retourne une valeur entière longue . La valeur retournée doit être entre les valeurs suivantes: - 9 223 372 036 854 775 807 et + 9 223 372 036 854 775 807 ou bien une erreur de dépassement ?OVERFLOW ERROR se produira. La valeur retournée est arrondie à l'entier + 1 si arg > .5 ou à l'entier - 1 si arg < .6. La valeur retournée peut être assignée à une variable entière longue.

**5** - CONV% évalue l'argument et retourne une valeur entière simple. La valeur doit être dans les limites de -32 768 à +32 767. La valeur retournée est arrondie à l'entier + 1 si arg > .5 ou à l'entier - 1 si arg < .6. La valeur retournée peut être assignée à une variable entière simple.

**6** - CONVà évalue l'argument et retourne une valeur entière double arrondie à l'entier supérieur. La valeur doit être dans les limites de -2 147 483 648 à +2 147 486 647. Un résultat inférieur à 0.5 sera arrondi à 0.

Exemple: CONV

```
10 A& = 123456 : B& = 23456
20 C& = CONV(A& - B&)
30 PRINT "Contenu de C&: "C&,"CONV(A& - B&): "CONV(A& - B&)
40 END

RUN
Contenu de C&: 100000          CONV(A& - B&): 100000
```

## Exemple CONV£

```
50 D& = 12345 : E& = 2345
60 F& = CONV(D& - E&)
70 PRINT "Contenu de F&: "F&,"CONV£(D& - E&): "CONV£(D& - E&)
80 END

RUN
Contenu de F&: 10000          CONV£(D& - E&): 10000
```

## Exemple CONV$

```
90  G% = 123 : H% = 23
100 I$ = CONV$(G% * H%)
110 PRINT "Contenu de I$: "I$,"CONV$(G% * H%): "CONV$(G% * H%)
120 END

RUN
Contenu de I$: 2829           CONV$(G% * H%): 2829
```

## Exemple CONV&

```
130 PRINT "Résultat de CONV&(300 - 1300): "CONV&(300 - 1300)
140 PRINT "Résultat de CONV&('1234567.987'): "CONV&("1234567.987")
150 END

RUN
Résultat de CONV&(300 - 1300): -1000
Résultat de CONV&('1234567.987'): 1234568
```

## Exemple CONV%

```
160 PRINT "Résultat de CONV%(123.45): "CONV%(123.45)
170 J% = 9876 : K% = 543
180 PRINT "Résultat de CONV%(J% / K%): " CONV%(J% / K%)"
190 END

RUN
Résultat de CONV%(123.45): 123
Résultat de CONV%(J% / K%): 18
```

## Exemple CONVà

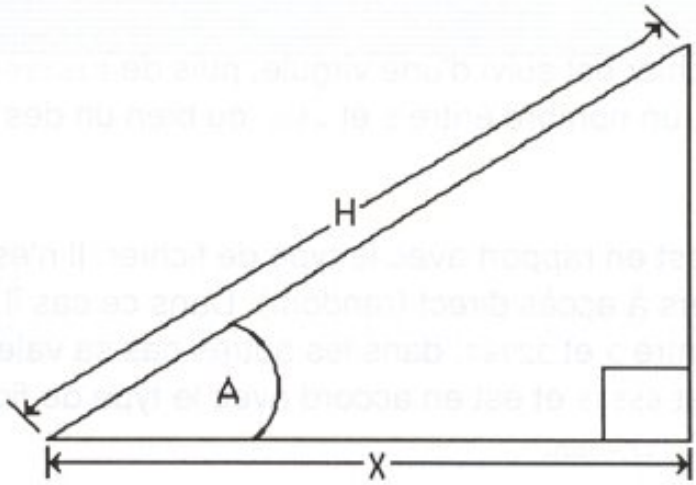
```
200 PRINT "Résultat de CONVà(123456.78): "CONVà(123456.78)
210 L = 9876.54 : M = 6543.21
220 PRINT "Résultat de CONVà(L / M): " CONVà(L / M)"
230 END

RUN
Résultat de CONVà(123456.78): 123457
Résultat de CONVà(L / M): 2
```

Token: DF AA

Syntaxe: `n = COS(A)`

La fonction `COS()` calcule le cosinus de l'angle A dont la valeur est exprimée en radians.

On note que pour une fois le résultat est correct et qu'Apple a épargné les classiques aberrations du genre `-0,99999` ou `-1,00001` que l'on a connues sur d'autres machines.




La fonction `COS()` se définit comme le rapport de la longueur du côté adjacent à l'angle `A` à la longueur de l'hypoténuse. Dans la figure ci-dessus, l'angle `A` sera défini comme suit: `A = COS(X/H)`

Exemple:

```
5  REM   COS.GSB
10 PRINT "COS(PI):       "COS(PI)
20 DEGRES = 180
30 RADIANS = (DEGRES * PI) / 180
40 PRINT "COS(RADIANS): "COS(RADIANS)
50 END

RUN

COS(PI):       -1
COS(RADIANS): -1
```

Cet exemple montre en ligne `10` le cosinus de `PI`. Ensuite, en lignes `20` à `40`, on a la conversion de `180` degrés en radians, puis son `COS`.

Token: A2 Commande

Syntaxe: `CREATE nomfichier, FILTYP=nnn [,rec]`

La commande `CREATE` permet de créer un fichier portant le nom `nomfichier` du type `FILTYP=nnn` et, dans certains cas, le `rec` ad hoc.

La syntaxe `du nomfichier` est celle de *ProDOS*:

- il doit commencer par une lettre,
- il comporte des lettres, des chiffres ou des points,
- il est composé de `15` caractères au maximum.

Le nom du fichier est suivi d'une virgule, puis de `FILTYP=nnn` avec, comme argument, un nombre entre `0` et `255`, ou bien un des codes mnémoniques.

L'option `rec` est en rapport avec le type de fichier. Il n'est obligatoire que pour les fichiers à accès direct (random). Dans ce cas il peut avoir une valeur comprise entre `3` et `32767`; dans les autres cas sa valeur est comprise entre `0` et `65535` et est en accord avec le type de fichier demandé par l'instruction `FILTYP=nnn`.

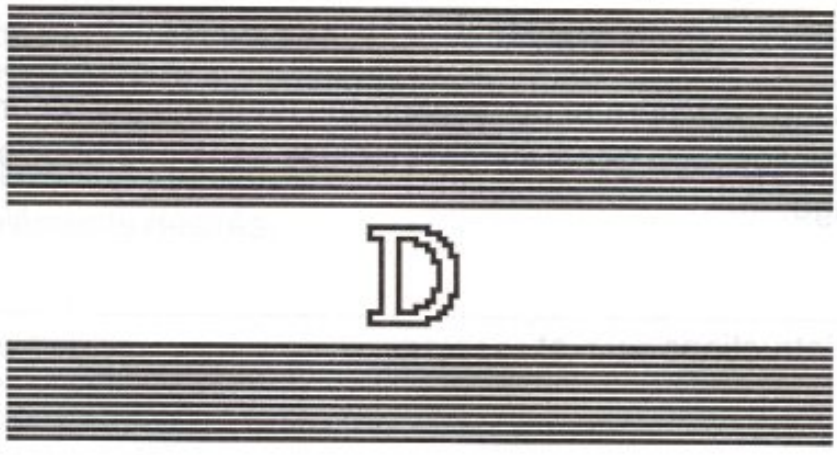
Si on tente d'entrer une valeur de sous-type dépassant `255`, ou si on tente de créer un fichier avec un nom existant, une erreur est affichée.

Exemple:

```
CREATE /GSB/BIN, FILTYP=6
CREATE /GSB/DIRECTORY, FILTYP=DIR
CREATE /GSB/TXT, FILTYP=TXT, 256
CREATE /GSB/GSB, FILTYP=CAT
```

Message d'erreur possible:

```
?BAD PATH ERROR
?DUPLICATE FILE ERROR
?ILLEGAL QUANTITY ERROR
?I/O ERROR
?VOLUME NOT FOUND
?DIRECTORY FULL
```




DATA..... 44
DATE()..... 45
DATE$..... 46
DBUG..... 48
DEF..... 49
DEL..... 51
DELETE..... 52
DIM..... 53
DIR..... 54
DIV..... 56
DO..... 57

Token: CA

Syntaxe:

```
DATA a[,b] [,c] [,"Zazou"] [,44] [,23%]
```

L'instruction DATA permet de stocker des données qui seront lues à partir de l'instruction READ généralement incluse dans une boucle de lecture/stockage.

Voir les intructions READ et RESTORE.

Exemple:

```
 10       REM    DATA.GSB
 20 DEBUT:PRINT "Démonstration de l'instruction DATA"
 30       DIM A$(8)
 40       FOR A%=0 TO 8
 50           READ A$(A%)
 60           PRINT A$(A%);"!";
 70       NEXT
 80 FIN:  END
 90       DATA "Ceci "," est "," la "," démonstration "
100      DATA " de "," l'instruction "," DATA "," à vous "," ... "

RUN

Démonstration de l'instruction DATA
Ceci ! est ! la ! démonstration ! de ! l'instruction ! DATA ! à vous !..!
```

Token: DF AF Fonction

Syntaxe: `a = DATE(n)`

La fonction `DATE()` est utilisée pour lire la date et l'heure de l'horloge incorporée de l'*Apple IIGS*. La valeur de l'argument `n` doit être comprise entre 0 et 4 en accord avec le tableau ci-dessous.

Il faut initialiser la fonction `DATE()` par `PRINT DATE(0)` avant de pouvoir récupérer les éléments désirés.

| Fonction | Description: |
|---|---|
| DATE(0) | Année -1900 (87 est retourné pour 1987) |
| DATE(1) | An |
| DATE(2) | Mois (entre 1 et 12) |
| DATE(3) | Numéro du jour (entre 1 et 31) |
| DATE(4) | Jour de la semaine (entre 1 et 7, Dimanche = 1) |

Exemple:

```
 10 REM              DATE.GSB
 20 An=0 : Mois=0 : NumJour=0 : JourSem=0 : Z=0
 30 DIM Mois$(12),Jour$(7)
 40 Mois$(1)="Janvier":Mois$(2)="Février":Mois$(3)="Mars"
 50 Mois$(4)="Avril":Mois$(5)="Mai": Mois$(6)="Juin"
 60 Mois$(7)="Juillet":Mois$(8)="Aout":Mois$(9)="Septembre"
 70 Mois$(10)="Octobre":Mois$(11)="Novembre":Mois$(12)="Décembre "
 80 Jour$(1)="Dimanche":Jour$(2)="Lundi":Jour$(3)="Mardi":Jour$( 4)="Mercredi"
 90 Jour$(5)="Jeudi":Jour$(6)="Vendredi":Jour$(7)="Samedi"
100 PRINT DATE(0) : HOME : REM     Initialise la fonction DATE()
110 An=DATE(1)
120 Mois=DATE(2)
130 NumJour=DATE(3)
140 JourSem=DATE(4)
150 PRINT:PRINT "Date: "Jour$(JourSem);" ";NumJour;" ";
160 PRINT Mois$(Mois);" ";An:PRINT
170 PRINT "La date vous a été fournie gracieusement par votre Apple favori!"
180 PRINT:END

RUN

Date: Mercredi 18 Novembre 1987

La date vous a été fournie gracieusement par votre Apple favori!
```

Token: DF E3

Syntaxe:

**1 -** `PRINT DATE$`
**2 -** `DATE$ an,mois,jour`

La première utilisation de DATE$ est la lecture de la date dans le format que vous avez choisi au *Tableau de Bord*.

Mode immédiat:

```
PRINT DATE$

18/11/1987
```

Mode différé:

```
10 PRINT DATE$
20 END

RUN

18/11/1987
```

La deuxième utilisation vous donne la possibilité de changer la date sans accès au tableau de bord.

Le format d'entrée est le suivant:

an = année - 1900
mois = nombre variant entre 1 et 12
jour = quantième du jour, varie entre 1 et 31 suivant le mois.

Le **IIGS Basic** refusera, sans afficher de message d'erreur, toute tentative d'entrée d'un numéro de jour ou de mois hors norme; par exemple: 31 pour un mois de Février, ou 13 comme numéro de mois. Par contre, tout numéro d'année supérieur à 99 fera apparaître des données fantaisistes.

Le changement de la date n'affecte en rien l'heure.

Mode immédiat:

```
DATE$ 87,11,19
```

fixera la date au 19 novembre 1987

Mode différé:

```
 10 REM       DATE.GSB
 20 PRINT "Date actuelle: "DATE$
 30 PRINT "Veuillez donner la date dans le format suivant AA,MM,JJ: "
 40 INPUT AA,MM,JJ
 50 DATE$AA,MM,JJ
 60 PRINT DATE$
 70 INPUT "La date est-elle correcte ? (O/N) ";OK$
 80 IF OK$="N" THEN 20
 90 PRINT "Au revoir...."
100 END

RUN

Date actuelle: 18/11/87
Veuillez donner la date dans le format suivant AA,MM,JJ:
?87
??11
??28
La date est-elle correcte ? (O/N) O
Au revoir....
```

Token: BA

Syntaxe:

Token: CD Instruction

Syntaxe:

```
1 - DEF FN nomfonction [car] (variable)
    [LOCAL variable1 [,variable2]]
    END FN nomfonction [car]

2 - DEF PROC nomprocedure [(variable)]
    [LOCAL variable1 [,variable2]]

    END PROC [nomprocedure]
```

L'instruction DEF est utilisée pour définir:

a - une fonction: DEF FN.
b - une procédure: DEF PROC.

Dans les deux cas, vous devez terminer la déclaration de fonction ou de procédure par:

a - END FN nomfonction [car]
b - END PROC [nomprocedure]

**A) Fonction:**

Le nom d'une fonction ne peut dépasser 29 caractères et a la même syntaxe que celui d'une variable; en outre, il ne doit déjà pas être utilisé. Il peut être suivi d'un des caractères optionnels suivants: %, à, & ou £.

Il doit y avoir au moins un paramètre entre parenthèses à la suite du nom de la fonction.

On peut écrire une fonction en n'utilisant qu'une ligne:

```
DEF FN Carre(x) = x * x
```

**B) Procédure:**

Le nom d'une procédure doit avoir au maximum 29 caractères et suivre la syntaxe des noms de variables. Il ne peut pas avoir été attribué dans le programme comme nom de procédure ou de fonction. Aucun caractère de type (%, à, &, £) ne doit se trouver à la fin du nom de procédure.

Le champ paramètre est optionnel, et peut contenir autant de paramètres (séparés chacun par une virgule) à concurence de 239 caractères sur la ligne. Avoir beaucoup de paramètres ralentira l'exécution de votre programme.

Il ne faut pas définir une fonction ou une procédure à l'intérieur d'une définition de procédure. Par contre, on peut appeler une fonction ou une autre procédure à l'intérieur de celle qui est définie.

Voir les deux mots réservés FN et PROC.

Exemple: FN

```
5  REM      DEF.FN.GSB
10  DEF FN Carre(cote)
20  Surfcarre = cote * cote
30  END FN Carre(cote)
40  INPUT "Valeur du coté: ";cote
50  x = FN (cote)
60  PRINT "Superficie:    "Surfcarre
70  END

RUN

Valeur du coté:  12
Superficie:     144
```

Exemple: PROC:

```
80  DEF PROC Initialisation
90  A$ = "Procédure d'initialisation" : B$ = " de chaines,"
100 C$ = " de variables entières ou non" : D$ = "Démonstration:"
110 E% = 1234 : F% = 44
120 END PROC Initialisation
130 PROC Initialisation
140 PRINT D$ : PRINT A$;B$;C$
150 PRINT "E% = "E%"F% = "F%
160 END

RUN

Démonstration:
Procédure d'initialisation de chaines, de variables entières ou non
E% = 1234        F% = 44
```

Token: 81 Commande

Syntaxe:

```
DEL n1,n2
DEL label1,label2
DEL n1-n2
DEL label1-label2
```

La commande DEL efface les lignes spécifiées en `n1,n2` ou `label1,label2` ou une combinaison entre ces deux possibilités.

DEL fonctionne **UNIQUEMENT** en mode immédiat.

Exemples:

```
DEL Debut,130
DEL 10,Message
DEL 20-50
DEL Debut,Milieu
```

Token: 9F Commande

Syntaxe:

```
DELETE nomfichier
```

La commande DELETE est utilisée pour effacer le fichier donné en argument. Un sous-catalogue ne peut être effacé que s'il est vide. Le nomfichier doit être un nom *ProDOS* valide.

Un certain nombre d'erreurs peuvent se produire si l'argument de DELETE n'est pas correct.

| Message d'erreur | Cause |
|---|---|
| ?VOLUME NOT FOUND ERROR | Le nom du volume n'existe pas |
| ?PATH NOT FOUND ERROR | Le sous-catalogue n'existe pas |
| ?FILE NOT FOUND ERROR | Le fichier n'existe pas |
| ?FILE LOCKED ERROR | Le fichier est verrouillé ou bien |
| | Le sous-catalogue n'est pas vide |
| ?WRITE PROTECTED ERROR | Le disque est protégé à l'écriture |
| ?FILE BUSY ERROR | Le fichier est toujours ouvert |

Exemple:

```
DELETE /GSB/SYSTEM/SYSTEM.SETUP/PANEL.SETUP
```

Efface le fichier PANEL.SETUP qui est dans les sous-catalogues SYSTEM.SETUP/, lui-même dans le sous-catalogue SYSTEM/ et qui sont tous deux contenus dans le volume /GSB/.

Token: AB

Syntaxe:

```
DIM variable[vartype] (n)
```

Pour allouer de la place à un ou plusieurs tableaux dans un programme, utilisez l'instruction DIM.

Si un nom de variable de tableau est utilisé sans l'instruction DIM, ce tableau est créé avec 11 éléments numérotés de 0 à 10. Passé l'élément numéro 10 ou si le nombre n est plus grand que le tableau, le message d'erreur ?BAD SUBSCRIPT ERROR est envoyé.

Essayer de dimensionner un tableau plus d'une fois, conduit au message d'erreur ?DUPLICATE DEFINITION. Cependant, on peut utiliser l'instruction ERASE pour effacer un tableau de manière à le redimensionner.

La taille maximum d'un tableau est de 32767 cellules, et celle d'une variable numérique est de 4096 Koctets.

Un type spécial de tableaux de variables (appelé structure ou tableau d'octets) peut être défini avec la commande DIM en utilisant le point d'exclamation (!).

Exemple:

```
  5         REM       DIM1.GSB
 10 DEBUT:  DIM Apple!(5),MacIntosh!(9)
 20         SET(Apple!(0))="Apple"+CHR$(13)
 30         SET(MacIntosh!(0))="MacIntosh."
 40 BOUCLE1:FOR i%=0 TO 9
 50              PRINT CHR$(MacIntosh!(i%));
 60         NEXT:PRINT
 70 BOUCLE2:FOR i% = 0 TO 5
 80              PRINT CHR$(Apple!(i%));
 90         NEXT
100         PRINT:PRINT
110 BOUCLE3:FOR i% = 9 TO 0 STEP -1: PRINT CHR$(MacIntosh!(i%));:NEXT
120 BOUCLE4:    FOR i% = 5 TO 0 STEP -1: PRINT CHR$( Apple!(i%));:NEXT
130 FIN:        PRINT:END
RUN

MacIntosh.Apple

.hsotnIcaM
elppA
```

Token: AA Commande

Syntaxe:

```
DIR [.Dn]
DIR [/GSB/SYSTEM]
DIR [/GSB=.C]
DIR [/GSB/,TDF]
```

La commande DIR affiche en 80 colonnes le contenu d'un disque ou d'un sous-catalogue spécifié par la chaîne argument. Il vous est possible de présélectionner le type de fichiers à afficher en utilisant une ou plusieurs des options suivantes:

| | |
|---|---|
| `DIR /GSB/=£` | Affichera tout nom de fichier se terminant par un chiffre, |
| `DIR =,-TXT` | Affichera tout nom de fichier qui n'est pas du type TXT, |
| `DIR =,DIR` | Affichera tout nom de fichier de type DIR. |
| `DIR =.GSB` | Affichera tout nom de fichier se terminant par .GSB |

Devant chaque nom de fichier peut apparaître un ou deux caractères *souris* (*Mouse Text*).

| Type | Code | Icône | Désignation |
|---|---|---|---|
| $04 | TXT | ▩ | Fichier texte ASCII |
| $06 | BIN | ≣ | Fichier binaire |
| $0C | SOS | ⌂ | Système d'exploitation Apple /// |
| $0F | DIR | 🗀 | Sous-catalogue |
| $AB | GSB | ◆ | Fichier programme GS Basic |
| $AC | BDF | ⊡ | Fichier data du GS Basic |
| $AD | TDF | ⌖ | Fichier librairie du GS Basic |
| $B3 | S16 | → | Fichier système P16 |
| $F9 | O.S | ⌘ | Système d'exploitation ProDOS 16 |
| $FF | SYS | ⌂ | Fichier système P8 |

Si aucun nom de volume ou de sous-catalogue ou .Dn n'est donné en argument, le contenu du préfixe actuel (préfixe 0) sera affiché.

Si la commande OUTPUT £n est utilisée avec n ayant une valeur autre que 0, le listing du catalogue sera envoyé au fichier correspondant à la valeur de n. La valeur de n sera comprise entre 1 et le nombre total de lecteurs de disque connectés à votre unité centrale. Voir la commande VOLUMES pour l'attribution exacte des numéros de périphérique sur votre configuration.

Le champ ACCESS définit les particularités d'accès au fichier. En effet, chacune des cinq lettres (D, N, B, W, R) définira, par sa présence ou son absence, la possibilité d'utiliser ou non chacun des privilèges.

Signification des lettres précitées:

D Possibilité d'effacer,
N Possibilité de renommer,
B Possibilité de copier,
W Possibilité d'écrire,
R Possibilité de lire.

DIR est également utilisé avec CREATE nomfichier, FILTYP= DIR pour créer un fichier du type sous-catalogue; notons que CREATE nomfichier, FILTYP= CAT donnera le même résultat.

Exemple:

```
DIR
/GSB/
   FILE NAME              TYPE  SIZE  LAST MODIFIED           ACCESS

 ⌂ PRODOS                  SYS    20K Tue, Jul 14, 1987 13:50 DNB WR
 🗀 SYSTEM                 DIR     1K Thu, Aou 27, 1987 12:49 DNB WR
 → GSBASIC.SYS16           S16    97K Fri, Sep 11, 1987 15:35 DNB WR
   FINDER.ROOT             $C9     1K Fri, Sep 11, 1987 15:59 DNB WR
   FINDER.DATA             $C9     1K Fri, Sep 11, 1987 15:59 DNB WR
 🗀 TDFS                   DIR     2K Fri, Sep 11, 1987 11:53 DNB WR
 🗀 ICONS                  DIR     1K Fri, Nov 13, 1987 11:47 DNB WR
 🗀 DEMOS                  DIR     1K Fri, Nov 13, 1987 11:46 DNB WR
 ◆ GSB.HELLO              GSB     1K Thu, Sep 10, 1987 20:42 DNB WR

 BLOCK FREE: 187  BLOCKS USED: 1413  TOTAL BLOCKS: 1600
```

Attention: Essayer d'envoyer DIR sur imprimante donne des résultats surprenants (caractères en double hauteur sur l'*Image Writer II* ).

Token: DF 8B Opérateur arithmétique

Syntaxe:

```
x = a DIV b
```

L'opérateur `DIV` effectue la division de ses arguments et retourne une valeur entière.

Exemple:

```
5  REM          DIV.GSB
10 A = 100 : B = 33 : X = 0
20 X = A DIV B
30 PRINT "Résultat de l'opération  '100 DIV 33' : "X
40 END

RUN

Résultat de l'opération  '100 DIV 33' : 3
```

Token: DC Verbe réservé

L'instruction DO définit le début d'une boucle DO .... UNTIL ou bien DO ... WHILE ... UNTIL.

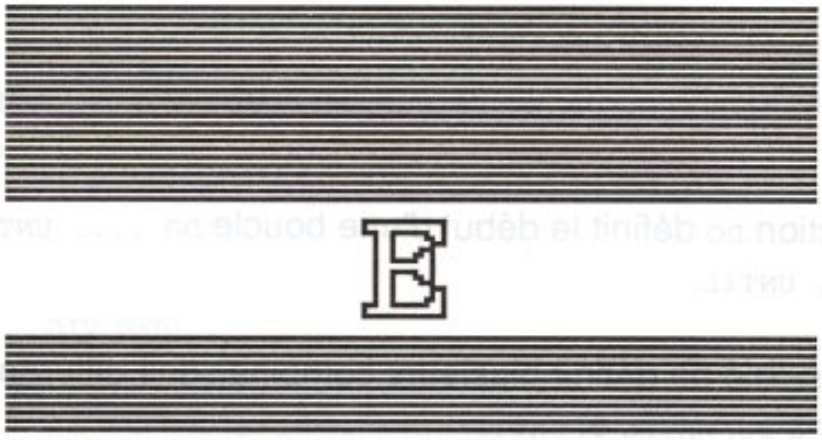
Il est possible de définir plusieurs combinaisons différentes à l'aide des instructions DO, WHILE et UNTIL.

| Combinaison | Appellation commune |
|---|---|
| DO: Instructions :UNTIL Condition | DO . UNTIL |
| DO: Instructions :WHILE Condition : Instructions :UNTIL | DO . WHILE . UNTIL |
| DO: Instruc :WHILE Condition : Instruc :UNTIL Condition | DO . WHILE . UNTIL |
| WHILE Condition : Instructions :UNTIL | WHILE . WEND |
| WHILE Condition : Instructions :UNTIL Condition | WHILE . UNTIL |
| WHILE : Instructions :UNTIL condition | REPEAT . UNTIL |
| WHILE : Instructions :UNTIL | Boucle infinie |

Se reporter aux instructions WHILE et UNTIL pour plus de précisions.

Messages d'erreur possibles:

```
?UNTIL w/o WHILE ERROR
?WHILE w/o UNTIL ERROR
```




EDIT..... 59
ELSE..... 61
END..... 62
EOF..... 63
EOFMARK()..... 64
ERASE..... 65
ERR..... 66
ERRLIN..... 67
ERROR..... 68
ERRTOOL..... 69
ERRTXT$()..... 70
EVENTDEF..... 71
EXCEPTION..... 72
EXEC..... 73
EXEVENTà()..... 74
EXFN..... 75
EXP(), EXP1(), EXP2()..... 76

Token: 82 Commande

Syntaxe:

```
EDIT
EDIT [numligne]
EDIT [label]
EDIT [numligne1] [,numligne2]
EDIT [numligne1] [,label]
EDIT [label] [,numligne1]
EDIT [label] [,label]
EDIT [TO £n]
```

La commande EDIT affiche une ligne en vue de la modifier. Il est possible de demander l'édition de plusieurs lignes en donnant le numéro de la première et de la dernière ou en panachant labels et numéros de lignes.

Le mode EDIT divise l'écran texte en deux parties: les quatre lignes du bas sont réservées à l'édition de la ligne courante et le reste de l'écran est utilisé pour stocker les lignes déjà éditées.

EDIT TO £n permet d'écrire les données dans le fichier correspondant à la valeur de n. Le fichier ouvert doit être du type *TXT* ou *SRC*, avoir été ouvert par la commande OPEN ..., FILTYP=... FOR OUTPUT AS £. ou bien OPEN ...,FILTYP= ... FOR APPEND AS £., ou être un fichier périphérique comme .PRINTER. Chaque ligne sera stockée dans une zone mémoire tampon de 512 octets avec un affichage à l'écran. Le texte sera écrit sur disque par tranche de 512 caractères entrés au clavier. Il faut évidemment avoir ouvert le fichier afin de lui donner un numéro de référence £n et le fermer de façon à sauvegarder les caractères en attente dans la mémoire tampon.

On ne peut pas utiliser EDIT TO £n en mode programme; aussi on se servira d'un fichier EXEC pour contourner la difficulté. Voir le programme en page suivante.

Il suffit de presser la touche <ESC> pour sortir de l'éditeur, que l'on soit en mode programme ou immédiat. Aucune distinction n'existe entre le mode immédiat et différé quand on utilise la commande EDIT TO £n. Chaque ligne tapée sera envoyée dans un tampon mémoire, puis dans le fichier n, même s'il s'agit de lignes **IIGS Basic**: il n'y aura aucune vérification de syntaxe et les mots ne seront pas tokenisés mais stockés comme du texte.

Le mode EDIT force le mode 80 colonnes et les caractères souris (*Mouse Text*) inactifs.

Exemple:

Un fichier EXEC du nom de EDIT.TO contient les lignes suivantes:

```
EDIT TO £1
CLOSE £1
```

Le programme ouvrira le fichier destination sous le numéro £1.

```
 10        REM   EDIT TO £1
 20 DEBUT:F$="EDIT.TO.SRC"
 30        OPEN F$, FILTYP= SRC AS £1
 40        PRINT "Tapez votre texte, puis pressez <RETURN> pour quitter"
 50        EXEC "EDIT.TO"
 60 FIN:   END
```

Token: 8D

L'instruction ELSE fait partie d'une boucle IF ... THEN ... ELSE. Si ELSE ne se trouve pas dans cette boucle, elle sera interprétée comme l'instruction REM.

Dans une construction IF test THEN instruction1: ELSE instruction2 la partie ELSE peut ne pas être sur la ligne IF ... THEN.

L'instruction ELSE ne sera exécutée que si la condition test est fausse.

Exemple:

```
 5  REM          ELSE.GSB
10 A = 1
20 IF A = 12 THEN B = 4
30 ELSE PRINT "A est différent de 12"
40 END

RUN

A est différent de 12
```

Token: DE

Syntaxe: END

L'instruction END (en français: FIN) signale à l'ordinateur la fin du programme en cours. END peut se trouver n'importe où dans un programme afin d'en arrêter l'exécution.

END diffère de STOP et de BREAK à deux égards:

1 - pas d'affichage de message d'interruption,
2 - fermeture de tous les fichiers.

L'utilisation de END est facultative; en effet, le programme s'arrête tout seul lorsqu'il n'y a plus de lignes et ferme tous les fichiers quand il a terminé son exécution.

Exemple:

```
10 PRINT "Voici un programme court!"
20 END

RUN

Voici un programme court!
```

Token: DF F1 Variable réservée

Syntaxe: `x = EOF`

Le **IIGS Basic** retourne le numéro de référence du fichier qui a causé l'erreur de fin de fichier dans la variable réservée EOF. Il est possible de tester la variable EOF pour connaître le fichier incriminé.

Si EOF est utilisée dans une boucle ON ... GOTO ou ON ... GOSUB, il faut la mettre entre parenthèses.

NB: EOF = fin du fichier (*End Of File*).

Exemple:

```
 10       REM    EOF.GSB
 20 DEBUT:F$="DATA.BDF":P$=".PRINTER"
 30       OPEN P$, AS £1
 40       OPEN F$, FOR INPUT AS £2
 50       FOR A%=0 to 20
 60          ON EOF£2 GOTO 100
 70          READ£2;Chaine$,Nombre
 80          PRINT£1;Chaine$,Nombre
 90       NEXT
100       CLOSE
110       PRINT "La fin du fichier a été rencontrée."
120 FIN:  END

RUN
```

Sur l'imprimante vous aurez le texte suivant:

```
Ligne 1
Ligne 2
Ligne 3
Ligne 4
Ligne 5
Ligne 6
Ligne 7
Ligne 8
Ligne 9
La fin du fichier a été rencontrée.
```

Token: DF B0 Fonction

Syntaxe: `x = EOFMARK(n)`

La fonction `EOFMARK()` retourne la grandeur en nombre d'octets du fichier référencé `n`. La valeur retournée tiendra dans une variable entière double.

Exemple 1:

```
OPEN MONFICHIER, FILTYP=TXT AS £1: PRINT £1 "EOFMARK(1) = "EOFMARK(1)
PRINT EOFMARK(1):CLOSE £1
16
```

La première ligne ouvre (et crée éventuellement) le fichier `MONFICHIER` de type `TXT` et lui donne `1` comme numéro d'identification; la deuxième écrit `EOFMARK(1) =` suivi du nombre d'octets contenu dans ce fichier, la dernière indique à l'écran la grandeur du fichier `MONFICHIER`.

Exemple 2:

```
 10       REM    EOFMARK.GSB
 20 DEBUT:F$="DATA.BDF"
 30       OPEN .PRINTER, AS £1
 40       OPEN F$, FOR INPUT AS £2
 50       FOR A%=0 to EOFMARK(2)
 60           READ£2;Texte$;Nombre
 70           PRINT£1;Texte$;Nombre
 80       NEXT
 90       CLOSE
100 FIN:  END

RUN
```

Sur l'imprimante, on aura le texte suivant:

```
Ligne 1
Ligne 2
Ligne 3
Ligne 4
Ligne 5
Ligne 6
Ligne 7
Ligne 8
Ligne 9
```

Exemple 3:

```
OPEN PICS, FILTYP=171 FOR INPUT £1 : PRINT EOFMARK(1)
6921
```

Token: D8

Syntaxe:

```
ERASE nomtableau() [,nomtableau()] [,nomtableau()]
```

ERASE efface aussi bien le tableau argument que les données qu'il contient. nomtableau est le nom du tableau à effacer. L'utilisation de ERASE libère l'espace mémoire utilisé par le tableau, la variable ou bien la structure d'octets et l'efface **DEFINITIVEMENT**. ERASE peut aussi être utilisé pour redimensionner des tableaux déjà attribués. Si on redéfinit un tableau avant d'utiliser ERASE on aura une erreur: ?DUPLICATE DEFINITION.

Pour effacer **TOUS** les pointeurs de tableaux, de variables et de variables de chaîne, il faut utiliser la commande CLEAR.

Exemple:

```
 10       REM ERASE.GSB
 20 DEBUT:DIM A$(15):B=15
 30       BREAK ON
 40       FOR I%=0 TO B
 50           A$(I%)=STR$(I%)
 60           PRINT A$(I%)"  ";
 70       NEXT
 80       PRINT:PRINT "ERASE A$()":ERASE A$()
 90       DIM A$(20):B=20
100       FOR I%=0 TO B
110           A$(I%)=STR$(I%)
120           PRINT A$(I%)"  ";
130       NEXT
140       PRINT:PRINT "Le tableau A$(15) a été redimensionné en A$(20)"
150 FIN:  END

RUN
0  1  2  3  4  5  6  7  8  9  10  11  12  13  14  15
ERASE A$()
0  1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20
Le tableau A$(15) a été redimensionné en A$(20)
```

Token: DF EF Variable

Syntaxe:

```
PRINT ERR
ON ERR GOTO sperreur
OFF ERR
```

Lorsque le **IIGS Basic** rencontre une erreur, il stocke son numéro dans la variable réservée ERR.

Il est possible d'utiliser ce numéro dans un programme pour déterminer quel genre d'erreur il s'est produit ou encore pour faire une routine de traitement d'erreurs. Un <CTRL-C> stoppera cettre routine car le <CTRL-C> est traité séparément par BREAK ON et BREAK OFF.

Tout ON ERR nécessite un OFF ERR. Chaque OFF ERR effacera l'effet du dernier ON ERR rencontré jusqu'au rétablissement du mode standard de traitement des erreurs.

Les instructions situées sur la même ligne et après le ON ERR seront traitées comme un commentaire.

Voir ERROR, ERRLIN, ERRTXT$ et également en annexe la liste des erreurs et leurs numéros.

Exemple:

```
 10       REM    ERR.GSB
 20 DEBUT:ON ERR GOTO SPTE
 30       INPUT "Entrez un nombre: ";A%
 40       PRINT "Vous avez entré le nombre: ";A%
 50 FIN:  END
 60 SPTE: OFF ERR : PRINT
 70       PRINT "Erreur n° "ERR " : "ERRTXT$(ERR)
 80       GOTO FIN

RUN
Entrez un nombre: 32768
Erreur n° 72 : SANE INVALID
```

L'erreur retournée s'explique car la valeur entrée au clavier est plus grande que ce que la variable A% peut recevoir; la valeur aurait dû être comprise entre -32768 et + 32767.

Token: DF EE Variable

Syntaxe: `PRINT ERRLIN`

Toute erreur qui se produit lors de l'exécution d'un programme se trouve dans une ligne dont le numéro est stocké dans la variable réservée `ERRLIN`. Associée à `ON ERR` et `ERRTXT$()`, cela peut faire une routine de traitement d'erreurs.

La variable `ERRLIN` ne renvoie que le numéro de ligne; il sera plus difficile de trouver l'erreur si plusieurs instructions sont sur la même ligne.

Voir `ERR`, `ERROR`, `ERRTXT$`, et également, en annexe, la liste des erreurs et leurs numéros.

Exemple:

```
 10       REM    ERR.GSB
 20 DEBUT:ON ERR GOTO SPTE
 30       INPUT "Entrez un nombre: ";A%
 40       PRINT "Vous avez entré le nombre: ";A%
 50 FIN:  END
 60 SPTE: OFF ERR : PRINT
 70       PRINT "Erreur n° "ERR " = "ERRTXT$(ERR)
 80       PRINT "En ligne "ERRLIN
 90       GOTO FIN

RUN
Entrez un nombre: 32768
Erreur n° 72 = SANE INVALID
En ligne 30
```

L'erreur retournée s'explique car la valeur entrée au clavier est plus grande que ce que la variable `A%` peut recevoir; la valeur aurait dû être comprise entre -32768 et + 32767.

Token: D7 Variable

Syntaxe: ERROR n

Il est possible à chacun d'utiliser ses propres définitions d'erreurs grace à la formule ON ERR.

La valeur de n sera comprise entre 0 et 255.

Si le numéro ne correspond pas à une erreur du **IIGS Basic**, le message d'erreur affiché sera celui-ci:
USER PROGRAM ERROR =nnn (où nnn représente le numéro d'erreur).

Voir ERR, ERRLIN, ERRTXT$ et également en annexe la liste des erreurs et leurs numéros.

Exemple:

```
ERROR 90
PROGRAM ERROR=$5A
```

Token: DF F5

Syntaxe:

```
ON ERRTOOL GOTO numligne
ON ERRTOOL GOSUB numligne
ON ERRTOOL GOTO label
ON ERRTOOL GOSUB label

PRINT ERRTOOL
x = ERRTOOL
```

Quand le **IIGS Basic** rencontre une erreur concernant les outils, il stocke le code erreur dans la variable réservée ERRTOOL.

On peut incorporer dans un programme une ou plusieurs routines de traitement des erreurs 'outils' en utilisant ON ERRTOOL. Une routine de traitement d'erreurs doit commencer par OFF ERRTOOL.

Un code d'erreur outil est composé de deux parties:

- l'octet de poids fort contient le numéro de l'outil,
- l'octet de poids faible contient le numéro de l'erreur.

Exemple:

```
?TOOL CALL ERROR=$0201
```

Outil n° $02 *Memory Manager,*
Erreur n° $01 *MemErr* : pas assez de mémoire disponible.

```
 10       REM Demo ERRTOOL
 20 DEBUT:ON ERR GOTO sptro:REM sous-programme de traitement d'erreurs outils
 30       LIBRARY APPEND "6/noteseq.tdf"
 40       REM Une erreur devrait se produire lors de
 50       REM    l'exécution de la ligne 30
 60       PRINT "Le chargement du fichier noteseq.tdf est terminé."
 70       END
100 SPTRO:OFF ERR
110       PRINT "Erreur n° $"ERRTOOL MOD 256;
120       PRINT " dans l'outil n° $"ERRTOOL DIV 256
130       END
```

**ATTENTION:** ON ERRTOOL n'est pas reconnu par la version 1.0B4.

Token: DF BF Fonction

Syntaxe: `PRINT ERRTXT$(n)`

La fonction `ERRTXT$()` retourne le texte correspondant à l'erreur `n`.

Si l'erreur `n` n'est pas une erreur du **IIGS Basic**, une chaîne nulle sera envoyée.

Voir `ERR`, `ERR`, `ERROR`, `ERRLIN`, et également en annexe la liste des erreurs et leurs numéros.

Exemple:

```
10        REM  ERRTXT.GSB
20 DEBUT:BREAK ON
30        FOR I%=1 TO 89
40            PRINT ERRTXT$(I%)
50 FIN:   END
```

Ce programme imprimera à l'écran le texte des messages d'erreur dont le code est compris entre 1 et 89.

Token: B1

EVENTDEF définit le numéro de départ des routines de prise en compte des événements (*Events* ) qui doit être appelé après que TASKPOLL ON a été exécuté.

Le **IIGS Basic** a une table interne de 64 événements. 27 d'entre-eux peuvent être retournés par le TASK MASTER. La table est indexée suivant le numéro de code de l'événement retourné par le TASK MASTER.

EVENTDEF définit le numéro de ligne à exécuter quand l'événement du TASK MASTER est retourné. L'index est un nombre entre 0 et 63 qui définit quel événement va être pris en compte dans le programme et le numéro de ligne de la routine qui l'exécutera.

Consultez dans le manuel de référence de la boîte à outils - dans la partie du *WINDOW MANAGER* - les définitions des numéros d'événements.

Les sous-programmes référencés par EVENTDEF doivent se terminer par RETURN 0. Cette forme spéciale de RETURN ne peut être utilisée que comme dernière instruction d'un sous-programme d'exécution d'événements TASK MASTER.

Token: 95

Syntaxe:

```
EXCEPTION ON n
EXCEPTION OFF
EXCEPTION 0
```

Il est possible de faire des opérations mathématiques à virgule flottante avec les routines mathématiques *SANE* . Le **IIGS Basic** permet au programmeur de contrôler les exceptions générées par l'outil *SANE*. Trois modes les prennent en compte: ils sont sélectionnés par l'instruction `EXCEPTION`.

**1 -** `EXCEPTION OFF` est le mode par défaut. Dans celui-ci, le **IIGS Basic** renvoie les messages d'erreurs standards pour les exceptions de calculs mathématiques importantes et ignore les exceptions mineures.

**2 -** `EXCEPTION 0` est utilisé pour déconnecter toutes les exceptions `SANE` et faire en sorte qu'elles soient ignorées et envoyées au résultat de l'expression et dans la variable réelle.

**3 -** `EXCEPTION ON` est utilisé pour permettre à un programme de prendre en charge toutes les exceptions.

Le nombre `n`, qui doit avoir une valeur comprise entre 0 et 63, est utilisé comme un masque pour filtrer les exceptions *SANE* . Ce masque détermine l'envoi ou non d'un message d'erreur **IIGS Basic**.

Lire à ce sujet le manuel: *Apple Numerics Manual* pour avoir une idée plus large de ce que sont les exceptions et de la manière de les utiliser.

Token: A5 Commande

Syntaxe:

```
EXEC /GSB/DIM.TXT [,OFF]
```

La commande EXEC permet de donner le contrôle à un programme de type texte qui peut contenir un fichier **IIGS Basic** qui sera seulement chargé, ou une suite d'instructions qui seront exécutées.

Si le fichier argument de EXEC n'est pas de type TXT ou SRC, une erreur sera envoyée: ?FILE TYPE MISMATCH.

L'option ,OFF supprime l'écho des caractères à l'écran.

La commande NEW, utilisée dans un fichier EXEC, ne sera pas exécutée. Dans ce cas, il faudra utiliser DEL 1-65279: CLEAR pour effacer le programme de la mémoire.

Il est possible d'utiliser EDIT £n ou PRINT £n pour créer un fichier EXEC à partir du **IIGS Basic**.

Pour passer des arguments au programme EXEC, on procède de la façon suivante:

syntaxe du fichier EXEC: `load é1è : REM chargera le programme argument,`
syntaxe d'appel: `exec fichier.exe,"pics: REM pics sera chargé!`

ATTENTION: l'instruction LIST doit se trouver sur la même ligne (dans le fichier EXEC) que OUTPUT £n et OUTPUT £n

Examinez, ci-dessous, le programme CAPTURE.

```
HOME
REM Transfert du programme é1è dans le fichier texte é2è
load é1è
open é2è, filtyp=TXT for output as £29
outrec=0 : indent =2
output £29 : list : output £0
outrec=80
close £29
type é2è
```

La syntaxe d'appel est: EXEC CAPTURE,"pics,pics.txt

Token: DF DC

Fonction

Syntaxe: `EXEVENTà(n)`

La fonction `EXEVENTà()` retourne une des 32 adresses des points d'entrées externes des événements.

L'adresse retournée peut être envoyée à la boîte à outils comme l'adresse d'une routine de gestion d'événement.

L'argument `n` est un nombre compris entre 32 et 63 et spécifie l'adresse de la routine de prise en compte d'événement.

Token: DF EB

Syntaxe: `x = EXFN_StringWidth("Ceci est un exemple")`

`EXFN_` exécute une fonction de la boîte à outils qui retourne une valeur numérique. Le nom de la fonction appelée doit se trouver dans le dictionnaire de librairie chargé par `LIBRARY` ou `LIBRARY APPEND`. `EXFN_` s'utilise de la même façon qu'une variable réservée; c'est-à-dire que l'on peut imprimer la valeur retournée en utilisant `PRINT EXFN_` ou bien attribuer à une variable la valeur retournée: `x = EXFN_`. Il est possible de faire suivre `EXFN` de l'un des cinq caractères spéciaux, (`%`, `à`, `&`, `$`, `£`), en fonction du type de valeur retrournée.

Pour récupérer l'adresse d'une variable numérique, il suffit d'utiliser la fonction `VARPTR()`. Les variables de chaîne sont converties en variables avec la longueur de la chaîne en premier caractère et l'adresse sera passée en argument.

La procédure standard des outils retourne le statut de l'erreur dans la retenue et le registre *A*.

Quand un appel à un outil est fait avec `EXFN_`:
1 - La retenue est à zéro: un zéro est retourné comme résultat de fonction.
2 - La retenue est à un: le contenu du registre *A* est retourné.

Message d'erreur possible:

```
?UNDEF'D PROC/FONCTION ERROR
```

Token: Fonction

**1 -** `DF A7` `EXP(`
**2 -** `DF A8` `EXP1(`
**3 -** `DF A9` `EXP2(`

**1 -** La fonction `EXP()` donne le nombre mathématique `e` (`e = 2.718282`) élevé à la puissance `n`. La valeur de l'argument `n` ne doit pas dépasser `11356,52340629`.

`EXP(2)` est plus précis que: `2.718282 ^ 2`. En effet, `EXP(2) = 7.389056`, alors que `2.718282 ^ 2 = 7.389057`

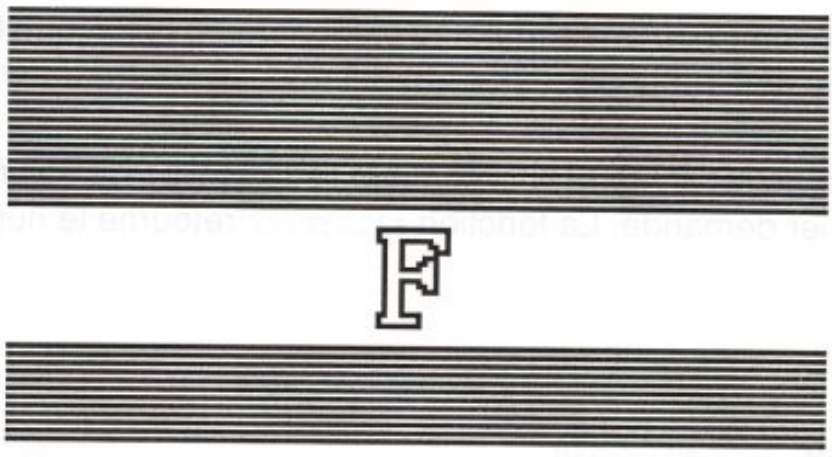
**2 -** La fonction `EXP1()` donne le nombre mathématique `e` ( `e = 2.718282`) élevé à la puissance `n-1`. La valeur de l'argument `n` ne doit pas dépasser `11356,52340629`.

`EXP1()` est plus précis que le calcul de `e puissance n -1` par exponentialisation et soustraction.

**3 -** La fonction `EXP2()` élève `2` à la puissance `n`. La valeur de l'argument `n` ne doit pas dépasser `16383.9999999`.

Message d'erreur possible:

```
?OVERFLOW ERROR
```




FILE() ..... 78
FILTYP() ..... 79
FILTYP= ..... 80
FIX() ..... 81
FN ..... 82
FOR ..... 83
FRE ..... 84
FREMEM() ..... 85

Token: DF DB Fonction

Syntaxe: `FILE("monfichier")`

L'instruction `FILE()` retourne une valeur 1 ou 0 selon que le fichier est présent ou non sur la disquette.

La variable réservée `AUXIDà` contient le sous-type du volume où se trouve le fichier demandé. La fonction `FILTYP(0)` retourne le numéro du type de fichier.

Exemple:

```
 10       REM      FILE.GSB
 20 DEBUT:INPUT "Nom du fichier à tester --> ";Fichier$
 30       ON FILE(Fichier$) GOTO 60
 40       PRINT "Le fichier "Fichier$" n'est pas dans le volume "PREFIX$
 50       END
 60       PRINT Fichier$" est présent dans le volume "PREFIX$
 70       PRINT PREFIX$Fichier$" a pour sous-type: "AUXIDà
 80       PRINT PREFIX$Fichier$" est du type: " FILTYP(0);
 90       PRINT " soit: $"RIGHT$(HEX$(FILTYP(0)),2)
100 FIN:  END

RUN

Nom du fichier à tester --> GSB.HELLO
GSB.HELLO est présent dans le volume /GSB/
/GSB/GSB.HELLO a pour sous-type: 0
/GSB/GSB.HELLO est du type 171 soit: $AB
```

Token: DF B1 Fonction

Syntaxe: `PRINT FILTYP(n)`

La fonction `FILTYP()` retourne le type de fichier d'un fichier ouvert. `n` est le numéro de référence attribué lors de son ouverture.

`FILTYP(0)` retourne le type du fichier appelé par la fonction `FILE`.

Exemple:

```
10      REM FILTYP.GSB
20 DEBUT:OPEN "Capture2", FILTYP= TXT AS £1
30      PRINT "Le fichier Capture2 est du type "FILTYP(1)"$"RIGHT$(HEX$
        (FILTYP(1)))
40      PRINT "TYPE Capture, TO £1 ...": TYPE "Capture", TO£1
50      CLOSE £1
60      PRINT "TYPE Capture2: ": TYPE "Capture2"
70 FIN:  END
```

Token: DF 92

Syntaxe: `FILTYP= type`

`FILTYP=` est utilisé lors de la création ou de l'ouverture d'un fichier pour fixer le type de fichier à créer.

`type` sera un nombre entre `0` et `255` ou bien le code mnémonique en trois lettres.

Exemple:

```
CREATE FICHIER, FILTYP=TXT
```

crée un fichier du nom `FICHIER` et de type `TXT`. On peut remplacer `FILTYPE=TXT` par `FILTYP=4`.

Token: DF B2 Fonction

Syntaxe: `x = FIX(n)`

La fonction `FIX()` renvoie l'entier de son argument `n`.

La différence entre `FIX` et `INT` réside dans le fait que `FIX` ne renvoie pas le nombre inférieur si `n` est négatif.

`x = FIX` est équivalent à l'expression:

```
x = SGN(n) * INT(ABS (n))
```

Exemple:

```
PRINT FIX(1.454)
1
```

Token: 87 Fonction

Syntaxe: `FN nom = expression`

La fonction `FN =` ou `FN LET` est utilisée à l'intérieur d'une définition multiligne d'une fonction ou d'une procédure.

La variable peut être:
- le nom d'une variable locale,
- une fonction ou un argument de procédure,
- le nom d'une fonction.

La variable doit être dans la table des variables locale sinon le message `?NOT LOCAL ERROR` sera affiché.

Cette restriction mise à part, `FN LET` fonctionne de la même façon que `LET`. Elle est normalement utilisée pour donner à une fonction la valeur résultante. Cependant, elle peut servir à vérifier que la valeur est bien donnée à l'intérieur d'une procédure ou d'une fonction locale et non globale.

Exemple:

```
  5 REM   FN.GSB
 10 DEF FN Rectangle(longueur,largeur)
 20 local S
 30 S = Longueur * Largeur
 40 FN Rectangle = S
 50 END FN Rectangle
 60 INPUT"Longueur du rectangle: ";long
 70 INPUT"Largeur du rectangle:  ";larg
 80 LET Surface = FN Rectangle(long,larg)
 90 PRINT "Surface du rectangle: "Surface
100 END

RUN

Longueur du rectangle: 10
Largeur du rectangle:  20
Surface du rectangle:  200
```

Token: 8B

Syntaxe:

```
FOR a% = b TO c [STEP d] : ... : NEXT [a%]
OPEN fichier, FOR INPUT AS £n
OPEN fichier, FOR OUTPUT AS £n
OPEN fichier, FOR APPEND AS £n
OPEN fichier, FOR UPDATE AS £n
```

L'instruction FOR associée à NEXT exécute un certain nombre de fois une série d'instructions. On appelle cette association *une boucle* .

`FOR a%` - est une variable entière qui sert de compteur,
`= b` - est une variable entière, valeur de début de la boucle,
`TO c` - est une variable entière, valeur de fin de la boucle,
`STEP d :` - argument optionnel, valeur entière de l'incrément à utiliser,
`NEXT [a%]` - le nom de la variable est optionnel. Si elle est présente, elle doit être identique à la variable qui suit FOR. S'il y a plusieurs `FOR ... NEXT`, l'ordre d'entrée des variables à la suite de FOR doit être rigoureusement respecté dans la composition de `NEXT var1,var2,var3` etc. Dans le cas où cette règle n'est pas suivie, le message `?NEXT w/o FOR ERROR` sera affiché.

Si STEP n'est pas utilisé, le pas par défaut est `1`. On peut demander un pas négatif ou positif.

Note importante: Utiliser une variable entière (`var%`) pour les boucles d'affichage.

```
FOR A  = 0 TO 32766: NEXT A: REM Temps d'exécution: 2 minutes 34 secondes
FOR A  = 0 TO 32766: NEXT  : REM Temps d'exécution: 2 minutes 23 secondes
FOR A% = 0 TO 32766: NEXT  : REM Temps d'exécution: --------> 23 secondes
```

Il est également possible d'utiliser FOR lors de l'ouverture d'un fichier. Suivant le mot réservé utilisé à la suite de FOR, on écrira dans le fichier (`FOR OUTPUT`), à la suite de ce fichier (`FOR APPEND`) ou bien on lira les données de ce fichier (`FOR INPUT`).

D'autre part, faites attention au débordement dans le cas des variables entières: en effet `FOR I%= 0 TO 32767` conduira à l'emballement de la machine. En effet, `I%` ne peut pas avoir une valeur supérieure à `32767`.

Token: DF ED Variable réservée

Syntaxe: `PRINT FRE`

Le **IIGS Basic** stocke dans la variable réservée `FRE` le nombre d'octets disponibles.

A chaque fois que `FRE` est appelée, le segment mémoire réservé aux chaînes est compressé de façon à libérer la place des anciennes chaînes inutilisées.

Exemple:

```
10 PRINT "Place disponnible actuellement: "FRE" octets"

RUN

Place disponnible actuellement: 32463 octets
```

Token: DF B3 Fonction

Syntaxe: `PRINT FREMEM(n)`

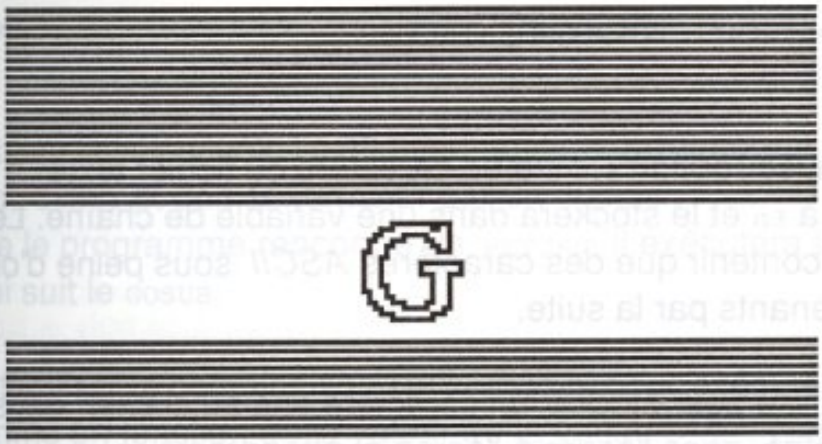
La fonction `FREMEM()` retourne des informations sur la mémoire disponible.

Voir le tableau page suivante.

Exemple:

```
LOAD DEMOS/PICS
PRINT "Taille du programmeactuel "FREMEM(4),"Taille du segment programme:
"FREMEM(5)

Taille du programme actuel 6921  Taille du segment mémoire: 8192
```

| n | Signification de la valeur retournée |
|---|---|
| 0 | Taille libre du segment de data avant compression des chaînes |
| 1 | Taille libre du segment de data après compression des chaînes |
| 2 | Taille de la mémoire allouée actuellement aux pointeurs. |
| 3 | Taille de la mémoire allouée actuellement aux variables simples, variables locales non comprises. |
| 4 | Taille du programme actuellement en mémoire. |
| 5 | Taille du segment de mémoire du programme. (x * 256 octets) |
| 6 | Taille du segment dictionnaire de librairie. |
| 7 | Taille totale de mémoire non allouée par le Memory Manager, et compacte la mémoire sans verrouiller le segment Basic. |
| 8 | Taille du plus grand block contigu libre. |
| 9 | Taille totale de la mémoire installée dans le système, sans compter les 64 K de mémoire réservés au générateur de son. |



GET ..... 88
GOSUB ..... 89
GOTO ..... 90
GRAF ..... 91

Token: D4

Syntaxe:

**1 -** `GET £n [,longueur] [,numrec] [;struct]`
**2 -** `GET$ var$`

**1 -** `GET` utilisé avec l'option `£n`, lira un caractère du fichier texte correspondant à `£n` et le stockera dans une variable de chaîne. Le fichier texte devra ne contenir que des caractères *ASCII* sous peine d'obtenir des résultats surprenants par la suite.

**2 –** `GET$` est utilisé pour stocker un caractère pris au clavier, dans une variable de chaîne, sans l'écrire à l'écran ni attendre un `<RETURN>`.

`GET$` traite `CTRL-C` comme n'importe quel autre caractère, le programme ne sera pas interrompu. `GET$` ne peut pas être utilisé avec une variable numérique.

**ATTENTION:** `GET$` retourne la variable avec une valeur supérieure à 127; je vous conseille donc d'utiliser la formule suivante:

```
GET$a$:a$=CHR$(ASC(a$)-128).
```

Exemples:

```
  5         REM     GET.GSB
 10 DEBUT:  DIM hello!(404)
 20         OPEN "hello.txt", FOR INPUT AS £1
 30         GET£1,404,0;hello!(0)
 40 BOUCLE:FOR i%=0 TO 404
 50             PRINT CHR$(hello!(i%));
 60         NEXT
 70 FIN:    END
  5         REM     GET1.GSB
 10 DEBUT:PRINT "Veuillez presser une touche S.V.P"
 20       PRINT "Control-à pour quitter ";
 30       GET$char$
 40       char$=CHR$(ASC(char$)-128)
 50       char=ASC(char$):PRINT
 60       IF char=127
 70           THEN PRINT "<DELETE> a été pressé: code ASCII: "char
 80       IF char<32
 90           THEN PRINT "Control-"CHR$(char+64)" code ASCII: "char
100           ELSE PRINT "Caractère: "char$" code ASCII: "char
110       IF char THEN 10
120 FIN:  END
```

Token: 89

Syntaxe:

```
GOSUB numligne
GOSUB label
```

L'instruction GOSUB envoie l'exécution au sous-programme qui débute en `numligne` ou `label` et doit se terminer par un RETURN.

Dès que le programme rencontre un RETURN, il exécutera la première instruction qui suit le GOSUB.

On peut utiliser la combinaison ON `var` GOSUB pour exécuter un sous-programme en fonction de la valeur de la variable `var`. Dans ce cas, les instructions qui suivent cette expression seront exécutées dès que le programme rencontrera un RETURN.

Plus de 40 GOSUB imbriqués les uns dans les autres donneront un débordement de pile. Le message d'erreur ?STACK OVERFLOW ERROR sera affiché.

Exemple:

```
  5 REM       GOSUB.GSB
 10 INPUT "Pressez un nombre compris entre 1 et 5 --> ";A
 20 ON A GOSUB 100,200,300,400,500
 30 ON A>5 GOTO 10
 40 END
100 PRINT "Ici la ligne 100":RETURN
200 PRINT "Ici la ligne 200":RETURN
300 PRINT "Ici la ligne 300":RETURN
400 PRINT "Ici la ligne 400":RETURN
500 PRINT "Ici la ligne 500":RETURN

RUN

Pressez un nombre compris entre 1 et 5 --> 5
Ici la ligne 500
```

Token: 8A

Syntaxe:

```
GOTO numligne
GOTO label
```

L'instruction GOTO fait suivre l'exécution du programme à la ligne ou au label spécifié. Il est possible de s'en servir en mode immédiat pour commencer l'exécution d'un programme à un point donné.

On peut utiliser l'expression ON var GOTO pour faire exécuter une instruction ou une suite d'instructions en fonction de la valeur d'une variable (var). Dans le cas où la variable var a une valeur nulle, les instructions qui suivent le ON var GOTO seront exécutées.

Attention: GOTO est une instruction à utiliser avec parcimonie car elle rend la lecture d'un programme très hardue.

Exemple:

```
  5 REM       GOTO.GSB
 10 INPUT "Pressez un nombre compris entre 1 et 5 --> ";A
 20 ON A GOTO 100,200,300,400,500
 30 ON A>5 GOTO 10
 40 END
100 PRINT "Ici la ligne 100"
200 PRINT "Ici la ligne 200"
300 PRINT "Ici la ligne 300"
400 PRINT "Ici la ligne 400"
500 PRINT "Ici la ligne 500"
510 END

RUN

Pressez un nombre compris entre 1 et 5 --> 2
Ici la ligne 200
Ici la ligne 300
Ici la ligne 400
Ici la ligne 500
```

Token: B9

Syntaxe:

```
GRAF INIT n
GRAF OFF
GRAF ON
```

GRAF ON provoque l'affichage de l'écran Super Haute Résolution. GRAF OFF permet de revenir à l'écran texte.

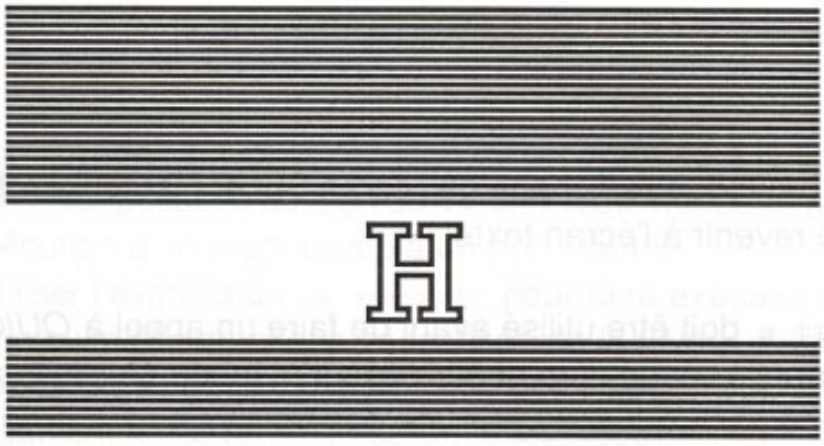
GRAF INIT n doit être utilisé avant de faire un appel à *QUICKDRAW II* de façon à allouer la page zéro et à initialiser le mode *QuickDraw* . Le mode est donné par n qui doit avoir une des trois valeurs suivantes: 0, 320 ou 640. GRAF INIT n est immédiatement suivi par un GRAF OFF généré par le systême. Il faudra donc envoyer un GRAF ON pour afficher l'écran SHR.

Vous pouvez passer du mode 320 au mode 640 et inversement. Dans ce cas, et si le *Window Manager*, *Menu Manager* et *Control Manager* sont activés, *QuickDraw* informera chaque outil du changement. GRAF INIT 0 indique que *QuickDraw* ne doit pas être réactivé.

L'exécution de GRAF INIT n initialise correctement *Quick Draw II* et le *Sound Manager* mais ni *NOTESYN* ni *NOTESEQ* .

Exemple:

Tapez GRAF ON en mode immédiat dès que vous accédez au **IIGS Basic** et vous verrez apparaître à l'écran l'image contenue dans la buffer SHR. Dans ce cas, il s'agira de l'écran du FINDER. Pour revenir à l'écran texte tapez: GRAF OFF ou bien TEXT.



HEX$() ..... 93
HLIST ..... 94
HOME ..... 95
HPOS ..... 96

Token: DF BC

Fonction

Syntaxe: `PRINT HEX$(n)`

La fonction `HEX$()` fournit une chaîne de 8 caractères qui représente la valeur hexadécimale (base 16) de l'argument décimal `n`.

La valeur de `n` doit être entre `+/- 2^ 32-1`, sinon une erreur `?ILLEGAL QUANTITY ERROR` sera affichée.

Exemple:

```
PRINT "80 Décimal = "HEX$(80)" Hexa"
80 Décimal = 00000050 Hexa

PRINT "90 Décimal = $"RIGHT$(HEX$(90),2)" Hexa"
90 Décimal = $5A Hexa
```

Token: 83 Commande

Syntaxe:

```
HLIST [label]
HLIST [label] [,label]
HLIST [numligne] [,label]
HLIST [label] [,numligne]
HLIST [numligne]
```

Efface l'écran et liste le programme **IIGSBasic** en fonction de l'option choisie. `HLIST` ne peut être utilisé qu'en mode immédiat.

Voir la commande `LIST`.

Token: BC Commande

Syntaxe: HOME

Efface l'écran et positionne le curseur dans l'angle haut/gauche de l'écran.

Exemple:

```
10 HOME
20 PRINT "Démonstration de HOME"
30 END
```

Token: DF E0 Variable réservée

Syntaxe: `HPOS = n`

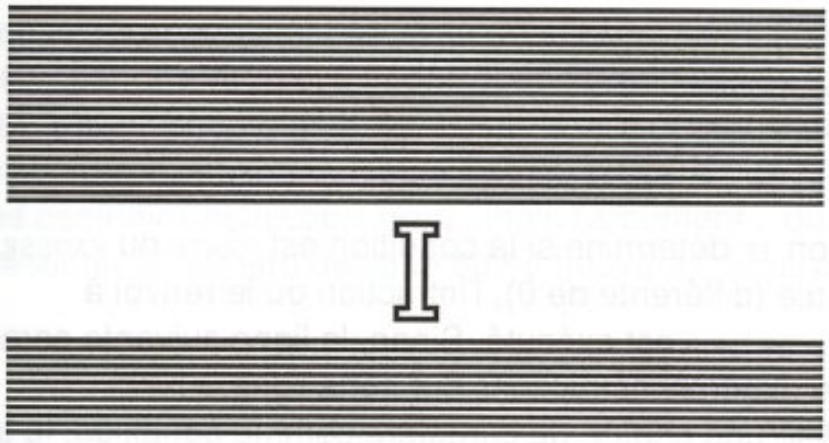
`HPOS` est une variable réservée qui contient la position horizontale actuelle d'écriture. Changer sa valeur modifiera la position d'écriture et celle du curseur qui peut être connue en se référant à la valeur de `HPOS`.

Si vous donnez à `HPOS` une valeur plus grande que 80, la position horizontale sera 80. Si cette valeur est plus grande que 255, le message d'erreur: `?ILLEGAL QUANTITY ERROR` apparaîtra.

Exemple:

```
HPOS = 35 : PRINT "Apple IIGS"
```

```
                                   Apple IIGS
```



IF..... 98
IMAGE..... 99
INDENT..... 102
INIT..... 103
INPUT..... 104
INSTR()..... 106
INT()..... 108
INVERSE..... 109
INVOKE..... 110

Token: C4

Syntaxe:

```
IF condition THEN instruction
IF condition THEN numligne
IF condition GOTO numligne
IF condition THEN label
IF condition GOTO label
```

L'instruction IF détermine si la condition est VRAIE ou FAUSSE. Si cette condition est vraie (différente de 0), l'intruction ou le renvoi à l'instruction en numligne est exécuté. Sinon, la ligne suivante sera analysée même s'il existe d'autres instructions sur cette ligne.

Si l'on utilise une chaîne de caractère comme condition, le test s'effectue sur la longueur de la chaîne. Si la chaine est nulle, elle est considérée comme fausse; vraie si elle contient un caractère.

Un numéro de ligne, un label ou une instruction peuvent suivre THEN .

La suite IF ... THEN ... ELSE peut figurer sur une, deux ou trois lignes différentes.

Exemple 1:

```
10 INPUT "Veuillez entrer un nombre SVP: ";A
20 IF A THEN PRINT "A est différent de 0"
30 IF NOT A THEN PRINT "A est égal à 0"
40 END

RUN
Veuillez entrer un nombre SVP: 10
A à une valeur de 0
```

Exemple 2:

```
10 A = 10 : B = 20 : C = 30 : D = 40
20 IF A = B
30 THEN PRINT "C: "C
40 ELSE PRINT "D: "D
50 END

RUN
D: 40
```

Token: CB

Syntaxe: `IMAGE spec1 [,spec2] [,spec3]`

L'instruction `IMAGE` contient une suite de spécifications (`spec1 ... specn`) séparées par une virgule. Chaque spécification correspond à une expression qui sera affichée par `PRINT USING`. N'utiliser `IMAGE` qu'à l'intérieur d'un programme. `IMAGE` doit être la seule instruction de la ligne. Plusieurs lignes peuvent contenir l'instruction `IMAGE`, mais l'argument `n` du `PRINT USING n;` devra représenter le numéro de ligne ou se trouvera l'`IMAGE` à utiliser.

Exemple:

```
 5 REM  IMAGE1.GSB
10 IMAGE 15A,24C,9R
20 IMAGE 10A,24C,9R
30 IMAGE 24A,24C,9R
40 PRINT USING 10;"J'utilise ici","l'IMAGE spécifiée à la","ligne 10."
50 PRINT USING 20;"Ici, c'est","l'IMAGE spécifiée à la","ligne 20."
60 PRINT USING 30;"Et enfin, ici nous avons","l'IMAGE spécifiée
   à la","ligne 30."
70 END

RUN
J'utilise ici  l'IMAGE spécifiée à la ligne 10.
Ici, c'est l'IMAGE spécifiée à la ligne 20.
Et enfin, ici nous avons l'IMAGE spécifiée à la ligne 30.
```

Les spécifications se divisent en trois groupes:

- spécification de chaîne (**1**), contrôle le format de l'affichage de la chaîne par `PRINT USING`,

- spécification littérale (**2**), insére un ou plusieurs espaces, un avancement de ligne, ou la copie d'une chaîne dans le texte affiché par `PRINT USING`,

- spécification numérique ( **3** ), contrôle le format d'une valeur numérique affichée par `PRINT USING`. On peut spécifier où se trouvera le point, la notation scientifique et la notation ingénieur.

**1 -** spécifications de chaîne:

La spécification de chaîne définit une fenêtre ayant `n` caractères de large et la façon d'afficher le texte à l'intérieur: `A` pour la justification à gauche, `R` pour la justification à droite ou `C` pour centré.

Soit: `IMAGE 15A,10R,25C`

Le premier argument de `PRINT USING` sera justifié à gauche et aura 15 caractères maximum affichés, le second sera justifié à droite avec 10 caractères maximum affichés et le troisième sera centré avec 25 caractères maximum affichés.

**2 -** spécifications littérales:

Une spécification littérale ne formate pas la valeur d'une expression; elle insère `n` espaces par `X`, insère `n` saut de ligne par `/`, écrit `n` fois une chaîne de caractères.

**3 -** spécifications numérique

Une spécification numérique formate l'affichage d'une valeur numérique en fonction de son type. Les trois types de formats numériques utilisent les caractères spéciaux suivants:

`£` réserve une position pour un caractère numérique et enlève les éventuels zéros en début du nombre,

`&` réserve une position pour un chiffre ou une virgule; un minimum de cinq chiffres doivent être réservés à la gauche du point décimal.

`Z` réserve une position pour un caractère numérique et écrit les éventuels zéros en début du nombre.

a) Dans ce format, *fixspec* , on peut utiliser les caractères spéciaux suivants:

`+` réserve un espace pour afficher le signe du nombre,
`-` réserve un espace pour afficher le signe moins, si le nombre est négatif,
`$` réserve un espace pour le signe dollar (`$`),
`**` affiche des astérisques à la place des espaces de tête,
`++` réserve les positions les plus à droite du nombre pour écrire le signe du nombre et le signe dollar (s'il est spécifié en en-tête),
`--` identique à `++`, sauf que le signe n'est écrit que si le nombre est négatif,
`$$` réserve les positions inutilisées les plus à gauche pour écrire le signe dollar et le signe du nombre (s'il en a un).

A noter: si `Z` est utilisé, n'employer ni `-`, ni `++`, ni `$$`. Le caractère spécial `**` doit figurer en premier dans la liste des spécifications et ne peut pas être utilisé avec `Z`, ce dernier enlevant tous les espaces inutilisés.

b) Format spécification scientifique: *scispec*

On a la possibilité d'envoyer les données dans la notation scientifique. Le format *scispec* est plus simple d'emploi que le format *fixspec*; en effet, il ne compte qu'un seul chiffre (ou pas de chiffre) avant le point décimal. Le nombre de chiffres à droite du point décimal se définit par `£` ou `n£`. La lettre `E` définit la position de l'exposant; on ne peut utiliser que `3E` ou `4E`, mais il est préférable de s'en tenir à `4E`. Le signe du nombre ne sera écrit que s'il y a assez d'espaces pour le faire!

c) Format spécifications ingénieur: *engrspec*

Ce format - très semblable au format *scispec* - force la valeur de l'exposant à être un multiple de trois et compte un maximum de trois espaces à gauche du point décimal.

Token: DF E7 Variable réservée

Syntaxe: `INDENT=n`

`INDENT` est une variable réservée qui contient le nombre d'espaces à rajouter pour indenter les boucles `FOR ... NEXT` dans les lignes de programme. Sa valeur par défaut est `2`.

Exemple:

```
INDENT=4
LIST

 5 REM     INDENT.GSB
10 REM Vous devez taper en mode immédiat: INDENT = n.
15 REM n représente le nombre d'espaces à utiliser pour l'indentation.
20 FOR A = 1 TO 20
30     PRINT "* * ";
40 NEXT
50 END

RUN

* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

Token: 98 Commande

Syntaxe: `INIT .Dn,/nomdisque`

La commande `INIT .Dn,/nomdisque` est utilisée pour formater un volume.

`.Dn` représente la position du lecteur dans la chaîne des lecteurs.
`/nomdisque` est le nom *ProDOS* de la disquette formatée.

Si la disquette est déjà formatée, le message suivant apparaît:

```
Press Y if you want to destroy /nomdisque?
```

répondre `Y` pour initialiser le disque, sinon rien ne se passe. Avec la version 1.0B4 vous ne pouvez initialiser que des disquettes vierges en mode immédiat.

En mode différé, le système n'effectue aucun contrôle: c'est au programme de réaliser les tests et d'avertir l'utilisateur.

Introduire une disquette formatée pour un autre système d'exploitation que *ProDOS (Pascal, Dos 3.3, HSF - Macintosh -)* conduit au message: `?VOLUME TYPE ERROR`. La commande n'est évidement pas exécutée.

Token: 91

Syntaxe:

**1 -** `INPUT [chaine] ;var1 [,var2]`
**2 -** `INPUT £numreference [,numenregistrement] ;var1 [,var2]`
**3 -** `INPUT USING numligne ; varchaine [,var1] [,var2]`

## 1 - INPUT

Il s'agit là d'un `INPUT` classique. L'élément `chaine` est optionnel, mais doit être suivi d'une virgule ou d'un point-virgule; dans ce cas, on n'a pas de `?` devant le curseur. On peut également faire un `INPUT` de plusieurs variables.

Exemple:

A - mode immédiat:

```
INPUT "Variable A,B,C,D: ";A,B,C,D
1,2,3,4

PRINT A" "B" "C" "D
1 2 3 4
```

B - mode différé:

```
10 INPUT "Veuillez entrer les nombres séparés par une virgule: ";An,Mois,Jour
20 PRINT "Jour: "Jour" Mois: "Mois" An: ";An
30 END

RUN

Veuillez entrer les nombres séparés par une virgule: 87,11,29
Jour: 29 Mois: 11 An: 87
```

## 2 - INPUT £n

La fonction INPUT £n lit une ligne de texte d'un fichier et lit les variables de gauche vers la droite. La ligne de texte peut se terminer par le caractère *ASCII* <CR> mais ce n'est pas obligatoire. INPUT £n lira les 255 caractères du fichier s'il ne trouve pas de CR. Le contenu du texte doit concorder avec le type de variable qui suit le INPUT £n. Chaque variable doit être séparée de la suivante par une virgule. Si la première ligne lue ne suffit pas à remplir les variables, une deuxième sera lue jusqu'à ce que toutes les variables de l'input soient utilisées, ou que la fin du fichier soit atteinte (EOF).

## 3 - INPUT USING

INPUT USING n exécute la *User Input Routine* en utilisant les paramètres de l'instruction IMAGE. La *User Input Routine* est la même routine qui est utilisée par la commande EDIT.

INPUT USING 0;var$ exécute un INPUT en utilisant les paramètres de la dernière instruction IMAGE utilisée.

*User Input Routine* : Routine d'input contrôlée par l'utilisateur.

Voir les instructions IMAGE et PRINT.

Token: DF D5 Fonction

Syntaxe: `INSTR("chaine1","chaine" [,n])`

La fonction `INSTR()` cherche la présence des caractères de la chaîne `chaine` dans la chaîne `chaine1`. L'argument optionnel `n` donne la position de départ de la recherche dans la chaîne `chaine1`. `INSTR()` retourne la position du premier caractère identique.

Si l'argument `n` a une valeur plus grande que la longueur de la chaîne, ou plus petit que `1`, on aboutit à l'erreur: `?ILLEGAL QUANTITY ERROR`.

Exemples:

Mode immédiat:

```
PRINT INSTR("Les feuilles mortes se ramassent à la pelle","se")
21
A = INSTR("All you need is love","love") : PRINT A
17
```

Mode différé:

```
  5 REM  INSTR.GSB
 10 HOME
 20 A$="Imagineheavenhell"
 30 B$="Imagine there's no heaven, it's easy if you try"
 40 C$="no hell below us, above us only sky"
 50 D$="Imagine all the people, living for today..."
 60 HPOS=36:PRINT LEFT$(B$,7):PRINT
 70 HPOS=27:PRINT LEFT$(B$,26)
 80 HPOS=30:PRINT RIGHT$(B$,20)
 90 HPOS=32:PRINT LEFT$(C$,16)
100 HPOS=31:PRINT RIGHT$(C$,17)
110 HPOS=29:PRINT LEFT$(D$,22)
120 HPOS=31:PRINT RIGHT$(D$,19):PRINT
130 PRINT "Les mots recherchés sont: <"LEFT$(A$,7)">  <"MID$(A$,8,6)"> <"
    RIGHT$(A$,4)">":PRINT
140 PRINT "Si le mot est trouvé, vous avez sa position";
150 PRINT " dans la chaine en début de ligne,"
160 PRINT "une ligne qui débute par 0 veut dire qu'il ";
170 PRINT "n'y a pas la chaine recherchée."
180 PRINT:PRINT INSTR(B$,LEFT$(A$,7));
190 HPOS=10:PRINT LEFT$(B$,7)
200 PRINT INSTR(B$,LEFT$(A$,7))","INSTR(B$,MID$(A$,8,6));
210 HPOS=10:PRINT B$
220 PRINT INSTR(C$,RIGHT$(A$,4));
230 HPOS=10:PRINT C$
240 PRINT INSTR("John Lennon - Imagine","Imagine",16);
250 HPOS=10:PRINT "John Lennon - Imagine"
260 PRINT:HPOS=20:PRINT "ooooO (c) 1971 Apple Records & EMI Ooooo"
270 END

RUN
                                Imagine

                        Imagine there's no heaven
                           it's easy if you try
                             no hell below us
                            above us only sky
                          Imagine all the people
                            living for today...

Les mots recherchés sont: <Imagine> <heaven> <hell>

Si le mot est trouvé, vous avez sa position dans la chaine en début de ligne,
une ligne qui débute par 0 veut dire qu'il n'y a pas la chaine recherchée.

1         Imagine
1,20      Imagine there's no heaven, it's easy if you try
4         no hell below us, above us only sky
0         John Lennon - Imagine

                    ooooO (c) 1971 Apple Records & EMI Ooooo
```

Token: DF 97 Fonction

Syntaxe : `PRINT INT(n)`

La fonction `INT(n)` fournit le plus grand nombre entier inférieur ou égal à `n`.

`n` est une expression numérique quelconque.

Une particularité de la fonction `INT(n)` est d'arrondir les nombres négatifs à la valeur supérieure (dans le sens négatif).

Voir par ailleurs la fonction `FIX`.

Exemples:

```
A = INT(44.32) : PRINT A
44

PRINT INT(-4.55)
5
```

Token: C1

Syntaxe: INVERSE

L'instruction INVERSE provoque l'affichage inverse en utilisant la couleur du fond pour l'écriture et la couleur de l'écriture pour le fond, telles qu'elles ont été définies au tableau de bord et suivant la configuration (moniteur couleur ou noir & blanc).

Token: 99

Syntaxe:

```
INVOKE
INVOKE nomfichier1 [,nomfichier2]
INVOKE APPEND nomfichier1 [,nomfichier2]
```

L'instruction INVOKE charge un fichier au format *Object Module Format* soit $B3 (S16). Le fichier argument de INVOKE sera localisé et chargé. Si le fichier n'est pas trouvé ou s'il n'est pas du bon type, une erreur est affichée. On peut charger plusieurs fichiers à la fois que vous voulez en séparant chaque nom de fichier par une virgule (à concurence de la mémoire disponible et du nombre de caractères maximum d'une ligne).

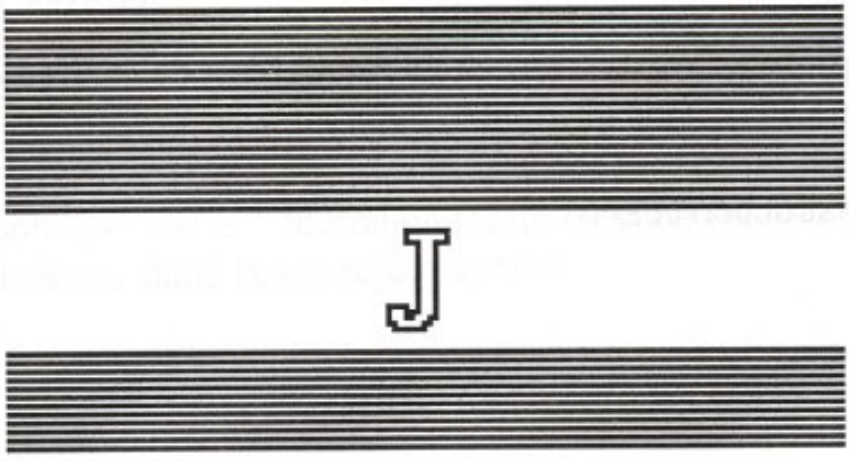
Exécuter INVOKE sans nom de fichier efface de la mémoire tous les fichiers chargés précédemment par un INVOKE; spécifier un ou plusieurs fichiers avec INVOKE efface également de la mémoire les modules chargés précédemment par un INVOKE mais charge ceux demandés. Pour ajouter un fichier à ceux déjà présents, utiliser INVOKE APPEND.

En mode différé, mettre entre guillemets le nom du fichier.

Les fichiers pouvant être chargés par INVOKE doivent avoir été écris en assembleur, en utilisant *APW* ou *MERLIN 16* suivant les spécifications de l'annexe I du manuel du **IIGS Basic**.

Messages d'erreur possibles:

```
?FILE NOT FOUND ERROR
?FILE TYPE ERROR
?SYNTAX ERROR
?OUT OF MEMORY ERROR
```




JOYY..... 112
JOYX..... 113

Token: DF F2 Variable réservée

Syntaxe: PRINT JOYY

L'instruction A = JOYX(n), fournit en retour l'état de l'entrée n dans A et l'état de l'entrée n+1 dans la variable réservée JOYY.

Voyez l'instruction JOYX().

Exemple:

```
A = JOYX(1) : PRINT " JOYY: ";JOYY
0
```

Token: DF 9B

Fonction

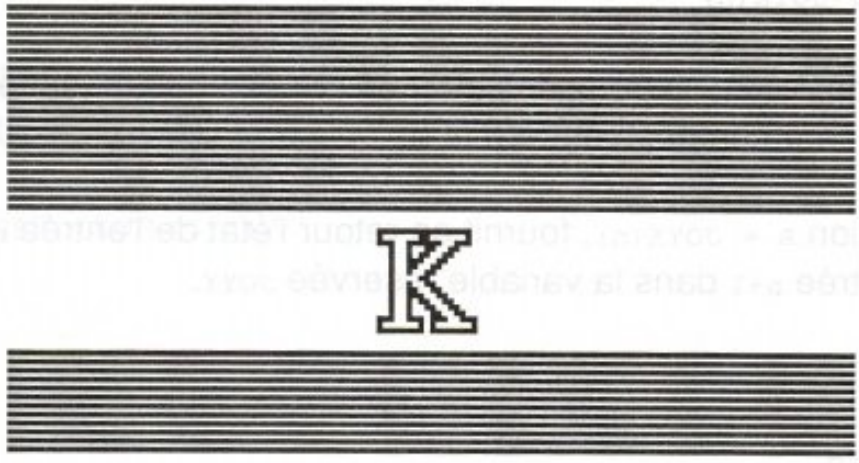
Syntaxe: `PRINT JOYX(n)`

La fonction `JOYX()` lit deux des quatres entrées des manettes de jeux. La valeur de `n` sera comprise entre 0 et 2.

L'instruction `A = JOYX(n)`, fournit en retour l'état de l'entrée `n` dans `A` et l'état de l'entrée `n+1` dans la variable réservée `JOYY`.

Exemple:

```
 5 REM JOYX.GSB
10 FOR i%=0 TO 2
20   PRINT "Etat de l'entrée "i%" --> ";
30   PRINT "  JOYX("i%"): "JOYX(i%)"    JOYY:  "JOYY
40 NEXT
50 END
```



KBD..... 115

## KBD

Token: DF F0

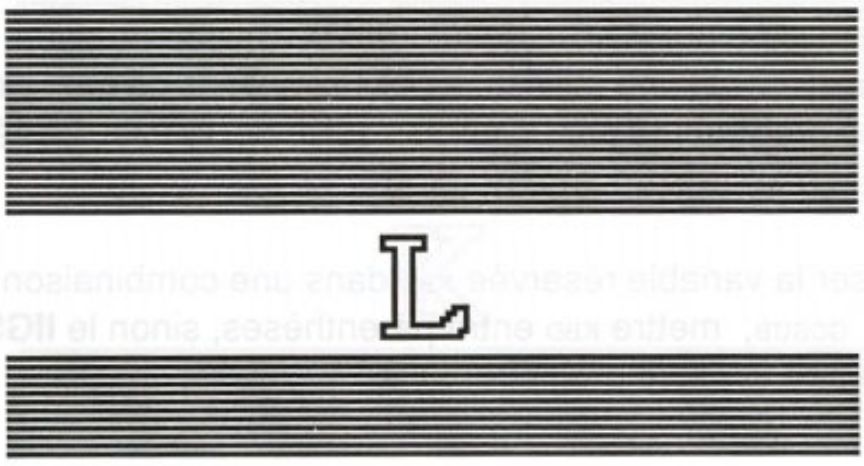
Syntaxe: `x = KBD`

KBD contient la valeur *ASCII* de la dernière touche pressée. ON KBD GOSUB ou ON KBD GOTO est une utilisation possible. Dans ce cas, il convient de terminer le sous-programme par un ON KBD suivit de RETURN. L'exécution d'un OFF KBD efface le dernier caractère pressé.

Pour utiliser la variable réservée KBD dans une combinaison ON ... GOTO ou ON ... GOSUB, mettre KBD entre parenthèses, sinon le **IIGS Basic** la traitera comme une variable ordinaire.

Exemple:

```
10 REM   KBD.GSB
20 PRINT "<Ctrl-à> pour stopper le programme"
30 ON KBD  GOTO 70:REM   Si une touche est pressée on va en 60
40 PRINT ".";:REM        En attendant, on imprime un point
50 GOTO 30:REM           On continue
60 END
70 PRINT KBD:ON KBD  GOTO 70
80 IF KBD=0 THEN 60:REM  Tant que KBD est différent de 0, on continue
90 RETURN
```




LEFT$() ..... 117
LEN() ..... 118
LET ..... 119
LIBFIND ..... 120
LIBRARY ..... 121
LIST ..... 122
LISTTAB ..... 123
LOAD ..... 124
LOCAL ..... 125
LOCATE ..... 126
LOCK ..... 127
LOG() ..... 128
LOG1() ..... 129
LOG2() ..... 130
LOGB%() ..... 131

Token: DF D1 Fonction

Syntaxe: `A$ = LEFT$(B$,n)`

La fonction `LEFT$()` fournit les `n` caractères à gauche de la variable `B$`. `B$` est une variable de chaîne; elle peut être aussi une chaîne de caractères. `n` est un nombre ou une variable numérique qui a une valeur comprise entre `1` et `255`.

Si on attribue à `n` une valeur `0`, la chaîne `A$` est nulle. Si on attribue à `n` une valeur plus grande que la longueur de la chaîne `B$`, la chaîne `A$` contient `B$`. Si la chaîne `B$` compte plus de `255` caractères, une erreur `?STRING TOO LONG` est affichée.

Exemple:

```
PRINT LEFT$("Bonjour",3)
Bon
```

Token: DF C6 Fonction

Syntaxe: `n = LEN(a$)`

La fonction `LEN()` retourne le nombre de caractères contenus dans la chaîne argument `a$`. L'argument `a$` peut être une variable de chaîne, une chaîne de caractères ou une concaténation de plusieurs chaînes.

La longueur de la chaîne peut être attribuée à une variable numérique.

Exemple:

```
PRINT LEN("IIGS Basic")
10
```

Token: C3 Mot réservé

Syntaxe: `LET variable = expression`

Le mot réservé `LET` permet d'attribuer la valeur d'une expression à une variable.

`variable` est le nom de la variable ou de l'élément de tableau qui recevra la valeur de l'expression.

`expression` est l'expression dont la valeur sera affectée à la variable ou à l'élément de tableau.

Le type de la variable doit être en accord avec celui de l'expression, sinon une erreur `?TYPE MISMATCH ERROR` est émise.

L'utilisation de `LET` est optionnelle, en effet `LET A = 12` et `A = 12` sont identiques.

Exemple:

```
LET A = 12 : PRINT A
12
```

Token: CC

Syntaxe: `LIBFIND "nomlibrairie",varentiere1%,varentiere2%,varentiere3%`

`LIBFIND` recherche dans le dictionnaire de la librairie, qui a été chargé avec l'instruction `LIBRARY`. Si le `nomlibrairie` n'est pas une chaîne nulle, le `nomlibrairie` est recherché dans la librairie. Le numéro de fonction est retourné dans `varentiere1`, le numéro d'outil est retourné dans `varentiere2` et `R.STACK` est retourné dans `varentiere3`. Si le `nomlibrairie` n'est pas trouvé, les trois variables entières sont mises à 0.

Exemple:

```
10 LIBFIND "QDStartUp",q1%,q2%,q3%
20 IF q1% = 0 THEN LIBRARY "/GSB/TDFS/QUICKDRAW.TDF"
30 REM Etc...
```

Token: 9A

Syntaxe:

```
LIBRARY nomlibrairie1 [,nomlibrairie2]
LIBRARY APPEND nomlibrairie1 [,nomlibrairie2]
```

L'instruction LIBRARY charge un ou plusieurs fichier de définition d'outils (en anglais: *Toolset Definition File* - TDF -)

S'il n'y a pas assez de mémoire libre pour charger tous les fichiers .tdf, une erreur: ?OUT OF MEMORY ERROR apparaîtra.

Il est possible de charger des fichiers .tdf avec l'instruction INVOKE. En utilisant LIBRARY sans APPEND, le dictionnaire de librairie est effacé, sauf les fichiers .tdf chargés par l'instruction INVOKE.

On peut charger en plusieurs lignes chaque fichier .tdf en utilisant: LIBRARY APPEND.

Exemple:

```
10 PREFIX 6,"/GSB/TDFS/"
20 LIBRARY APPEND "6/menu.tdf"
30 LIBRARY APPEND "6/control.tdf"
40 REM Etc...
```

Token: 84

```
LIST
LIST [,numligne]
LIST [numligne,]
LIST [numligne1] [,numligne2]
LIST [label] [,numligne]
LIST [numligne] [,label]
LIST [label] [,label]
```

Liste le programme actuellement en mémoire. Si une option est utilisée LIST affiche le texte comme suit:

```
,ndl ou -ndl                          ligne 0 à numéro de ligne,
,label ou -label                      ligne 0 à label,
ndl1,nl2 ou ndl1-ndl2                 ndl1 à ndl2,
ndl1,label ou ndl1-label              ndl1 à label,
label1,label2 ou label2-label2        label1 à label2,
label,ndl ou label-ndl                label à ndl.
```

Token: DF E9

Syntaxe: `LISTTAB = n`

`LISTTAB` est une variable réservée modifiable. Sa valeur lui est attribuée à chaque chargement d'un programme **IIGS Basic**. Elle peut être modifiée par l'utilisateur. Sa valeur par défaut est `5`.

Pour obtenir une indentation correcte des labels et des débuts de lignes, il faut lui donner une valeur de `6 + x` ( `x` = longueur du label le plus long).

Quand la valeur de `LISTTAB` est plus grande que `127`, les numéros de lignes ne sont pas affichés lors du `LIST` ou du `HLIST`.

Un listing d'un programme sans aucun numéro de ligne peut être affiché à l'écran, envoyé à l'imprimante ou stocké dans un fichier texte.

Token: 9D Commande

Syntaxe: `LOAD nomfichier`

La commande LOAD charge le fichier du type GSB spécifié par `nomfichier` en mémoire.

Toutes les variables du programme chargé sont mises à zéro pour les variables numériques et les chaînes de caractères mises en chaînes nulles. Tous les fichiers sont fermés sauf les fichiers EXEC. S'il y a déjà un programme en mémoire, il est effacé.

Utiliser LOAD pour charger un fichier de type autre que GSB conduit à une erreur.

Messages d'erreur possibles:

```
?I/O ERROR
?FILE NOT FOUND
?FILE TYPE ERROR
?PATH NOT FOUND
?VOLUME NOT FOUND
```

Token: D9

Syntaxe: `LOCAL nomvariable1 [,nomvariable2]`

L'instruction `LOCAL` ne peut être utilisée que dans la définition d'une fonction ou d'une procédure. Si `LOCAL` est employé ailleurs, dans un programme, une erreur sera générée. Les structures ne sont pas autorisées en `LOCAL`.

Une table des variables locales est créée chaque fois qu'une fonction est exécutée par `FN nom (...)` ou par `PROC nom(...)`. Les arguments des fonctions ou des procédures sont toujours insérés comme variables locales dans la table des variables locales.

Message d'erreur possible:

```
?NOT LOCAL ERROR
```

Token: B0 Fonction

Syntaxe: `LOCATE [ligne] [,colonne]`

`LOCATE` positionne le curseur de l'écran texte en `ligne` et `colonne` spécifiés par ses arguments.

L'argument `ligne` doit avoir une valeur comprise entre `1` et `24` et l'argument `colonne` une valeur comprise entre `1` et `80`. La valeur `colonne 0` est transformée en `1`.

Exemple:

```
LOCATE 2,30 : PRINT "Le ciel est bleu."
                              Le ciel est bleu.

LOCATE ,30 : PRINT "Les oiseaux chantent."
                              Les oiseaux chantent.

LOCATE 9 : PRINT "Et 'My tailor is rich'."

                              Et 'My tailor is rich'.
```

Token: A3

Syntaxe: `LOCK nomfichier`

`LOCK` interdit l'écriture ou l'effacement du fichier donné en argument. Lors du catalogue un astérisque signale les fichiers protégés à l'aide de `LOCK`.

Les noms de volumes ne peuvent pas être protégés à l'aide de l'instruction `LOCK`.

Voir `UNLOCK`.

Token: DF A3 Fonction

Syntaxe: `x = LOG(n)`

`LOG()` retourne le logarithme naturel (en base e) de son argument.

Exemple:

```
 5          REM      LOG.GSB
10          A = EXP(1)
15  BOUCLE: BREAK ON
20          FOR B% = 0 TO 5
30            PRINT LOG(A),A
40            A = A * A
50          NEXT
60          PRINT LOG(A),A
70          END

RUN

1           2.718282
2           7.389056
4           54.59814
8           2980.957
16          8886106
32           7.896288E+13
64           6.235136E+27
```

Token: DF A4 Fonction

Syntaxe: `x = LOG1(n)`

`LOG1()` retourne le logarithme naturel (en base e) de 1, plus la valeur de son argument.

Exemple:

```
 5         REM LOG1.GSB
10 DEBUT:  A=EXP1(1)
20 BOUCLE:BREAK ON
30         FOR i%=0 TO 15
40             PRINT LOG1(A),A
50             A=A+A
60         NEXT
70         PRINT LOG1(A),A
80 FIN:    END

RUN
1             1.718282
1.48988       3.436564
2.063455      6.873127
2.690989      13.74625
3.349641      27.49251
4.025084      54.98502
4.70926       109.97
```

Token: DF A5

Fonction

Syntaxe: `x = LOG2(n)`

`LOG2()` retourne le logarithme en base 2 de son argument.

Exemple:

```
 5         REM       LOG2.GSB
10         A = 2
20 BOUCLE:FOR B% = 0 TO 5
30              PRINT LOG2(A),A
40              A = A * A
50         NEXT
60         PRINT LOG2(A),A
70         END

RUN

1             2
2             4
4             16
8             256
16            65536
32             4.294967E+09
64             1.844674E+19
```

Token: DF A6

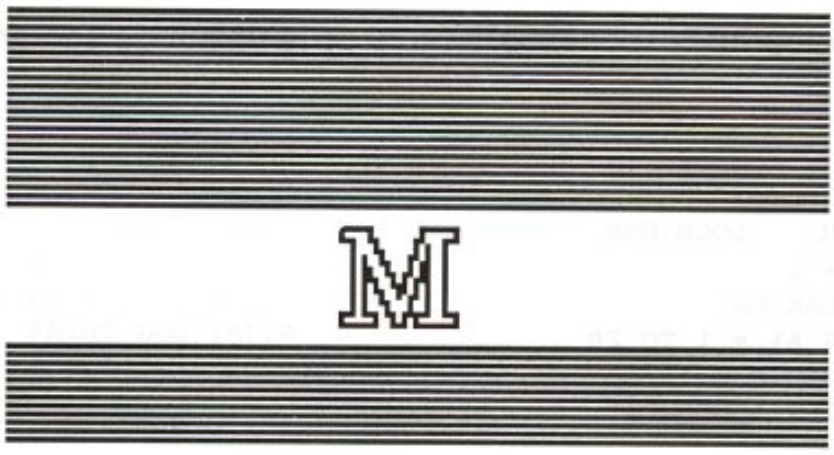
Fonction

Syntaxe: `x = LOGB%(n)`

`LOGB%()` retourne l'exponentiel binaire de la valeur de son argument.

Exemple:

```
 5          REM     LOGB.GSB
10          B = 1
15 BOUCLE:  BREAK ON
20          FOR A% = 1 TO 20
30              B = B * 2
40          PRINT LOGB%(B)"  ";
50          NEXT
60          END

RUN

1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20
```



MENUDEF..... 133
MID$()..... 134
MOD..... 135

Token: B2

Syntaxe:

```
MENUDEF index,numligne [,numligne]
MENUDEF index,numligne [,label]
MENUDEF index,label [,label]
MENUDEF index,label [,numligne]
```

MENUDEF définit le nom de la routine de prise en charge du menu à exécuter quand le *Task Master* retourne le numéro du menu choisi.

Les numéros d'identification définis par le *Menu Manager* doivent avoir une valeur comprise entre 256 et 383. Le **IIGS Basic** utilise le numéro d'identification moins 256 comme index dans sa table interne de numéro d'identification de menu.

MENUDEF est employé pour définir l'entrée dans la table interne. Le paramètre index sera un nombre entre 0 et 127 qui sélectionnera l'entrée à définir ou mise à 0. Si plusieurs lignes sont utilisées, chaqu'une d'elles définira la prochaine entrée dans la table d'identification de menu.

MENUDEF est seulement utilisable conjointement à TASKPOLL, avec le *Window Manager* et le *Menu Manager* .

Token: DF D4 Fonction

Syntaxe: `A$ = MID$(B$,n [,n1])`

La fonction `MID$()` permet d'extraire d'une chaîne, un nombre de caractères `n1` depuis le caractère `n`.

Si l'élément optionnel `n1` est plus grand que le nombre de caractères restant, ou s'il n'est pas présent, tous les caractères à compter de la position `n` seront retournés.

Exemple:

```
 5 REM MID.GSB
10 A$="TREMPLIN MICRO Editions JIBENA"
20 PRINT MID$(A$,16):REM Affiche la deuxième moitié de A$
30 PRINT MID$(A$,1,14):REM Affiche la première moitié de A$
40 END

RUN
Editions JIBENA
TREMPLIN MICRO
```

Token: DF 85 Opérateur

Syntaxe: `x = n1 MOD n2`

`MOD` (modulo) retourne dans `x` le résultat entier de la division de `n1` par `n2`. Cela évite toute une gymnastique obligatoire si on passe par `INT`.

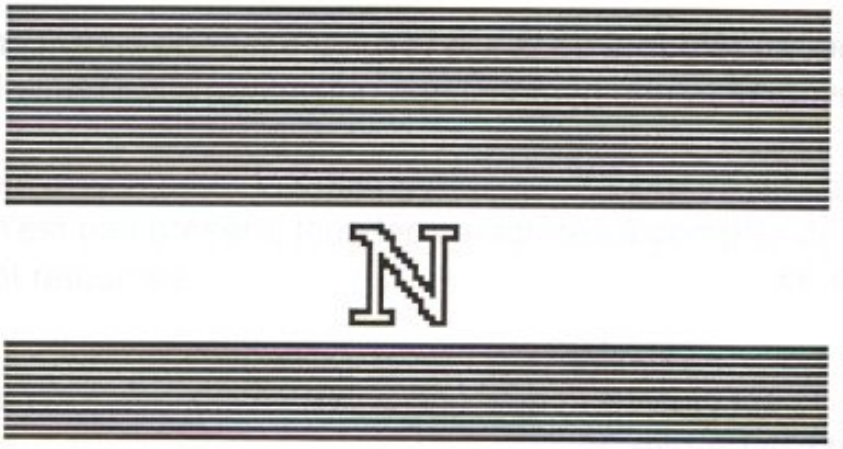
Exemple1:

```
PRINT 100 MOD 33
1
```

`100 MOD 33` est plus court que `100 - INT (100 / 33) * 33` et donne le même résultat c'est-à-dire: `1`.

Exemple2:

```
 5 REM MODULO.GSB
10 PRINT "    15 MOD 4 --> "15 MOD 4
20 A=15 MOD 4
30 PRINT "A = 15 MOD 4 --> "A
40 END

RUN
    15 MOD 4 --> 3
A = 15 MOD 4 --> 3
```



NEGATE..... 137
NEW..... 138
NEXT..... 139
NORMAL..... 140
NOT..... 141
NOTRACE..... 142

Token: DF B4 Fonction

Syntaxe: `PRINT NEGATE(n)`

`NEGATE()` retourne la valeur `-n`. Cette fonction a été incluse dans le **IIGS Basic** à cause de *SANE* .

Voir le manuel *Apple numerics manual - SANE -*.

Exemple:

```
 5 REM NEGATE.GSB
10 TEXT:HOME
20 PRINT "NEGATE i%         i%"
30 FOR i%=-8 TO 8
40   PRINT NEGATE(i%),i%
50 NEXT
60 END

RUN
8               -8
7               -7
6               -6
5               -5
4               -4
3               -3
2               -2
1               -1
0               0
1               1
2               2
3               3
4               4
5               5
6               6
7               7
8               8
```

Token: D1 Commande

Syntaxe: `NEW [n]`

`NEW` sans l'option `n` efface le programme actuellement en mémoire et toutes ses variables, ferme tous les fichiers ouverts à l'exception d'un fichier texte en cours d'exécution.

`NEW n` peut être utilisé pour réduire la grandeur du segment programme. En effet, le segment programme n'est pas réduit lors du chargement d'un programme plus court; c'est donc la seule façon de fixer sa grandeur en relation avec celle du nouveau programme.

La valeur de `n` doit être un multiple de `256` et doit être environ `256` octets plus grand que le programme.

Si on donne à `n` une valeur trop grande on aboutira à une erreur: `?ILLEGAL QUANTITY ERROR` ou bien `?OUT OF MEMORY`.

Exemple:

```
LOAD /GSB/DEMOS/PICS
PRINT FREMEM(4);: NEW 8448 :PRINT ,FREMEM(4)
8192            8448
```

Token: 8E

Syntaxe: `FOR var = n TO n1 : NEXT [var]`

`NEXT` fait partie de la suite d'instructions `FOR ... NEXT`. L'argument optionnel `var` permet d'agir seulement sur une boucle. Il faut signaler que `NEXT var` est plus lent à l'exécution que `NEXT`. D'autre part, utiliser des variables entières (`var%`) pour les boucles d'affichage car elles sont plus rapide.

Attention à bien spécifier `BREAK ON` avant d'entrer dans une boucle, sinon en cas d'emballement, on ne pourrait en sortir qu'en relançant le système.

Le **IIGS Basic** indente les lignes comprises entre les instructions `FOR` et `NEXT`.

Voyez `INDENT`.

Exemple:

```
FOR A  = 0 TO 32766 : NEXT A : REM Temps d'exécution: 2 minutes 34 secondes
FOR A  = 0 TO 32766 : NEXT   : REM Temps d'exécution: 2 minutes 23 secondes
FOR A% = 0 TO 32766 : NEXT   : REM Temps d'exécution:           23 secondes
```

Sans commentaire !

Token: C0

Syntaxe: NORMAL

L'instruction NORMAL rétablit l'affichage vidéo tel qu'il a été fixé au tableau de bord.

Voir INVERSE

Exemple:

```
  5 REM       NORMAL.GSB
10 PRINT "Exemple d'utilisation de l'instruction ";
20 INVERSE:PRINT "NORMAL";:NORMAL
30 END
```

Exécuter le programme pour visualiser l'effet de INVERSE et de NORMAL.

Token: DF 8D

Opérateur logique

Syntaxe: `PRINT NOT (val1 = val2)`

L'opérateur logique `NOT` retourne le résultat logique inverse de l'expression argument.

Une opération logique retournera `1` pour vrai et `0` pour faux; `NOT` retournera donc le contraire soit `0` pour vrai et `1` pour faux.

Exemple:

```
 5 REM NOT.GSB
10 A = 5 : B = 10 : C = 15 : D = 20
20 PRINT "NOT (A = B):      "NOT (A = B)
30 PRINT "NOT (A + C = D): "NOT (A + C = D)
40 PRINT "NOT (A <> B):     "NOT (A <> B)
50 IF NOT A THEN PRINT "IF NOT A:  A = "A:ELSE:PRINT "ELSE: A= "A
60 END

RUN

NOT (A = B):      1
NOT (A + C = D): 0
NOT (A <> B):     0
```

Dans le premier cas, le résultat est `1` soit faux, dans les deux autres cas, le résultat est `0` soit vrai.

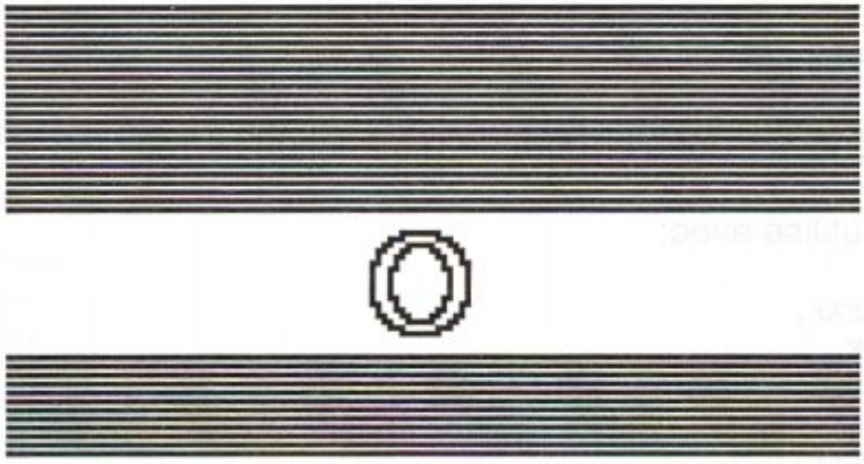
Token: BF

Syntaxe: NOTRACE

Arrête l'affichage du numéro de ligne accompagnant chaque instruction exécutée. NOTRACE annule l'effet de TRACE.

Exemple:

```
  5 REM   NOTRACE.GSB
10 TRACE
20 PRINT "Ici, nous sommes ";
30 PRINT "en mode 'TRACE'"
40 NOTRACE
50 PRINT
60 PRINT "Et ici, nous sommes en mode 'NOTRACE'"
70 END

RUN

   £20 Ici, nous sommes    £30 en mode 'TRACE'
   £40
Et ici, nous sommes en mode 'NOTRACE'
```



OFF..... 144
ON..... 145
OPEN..... 146
OR..... 147
OUTPUT..... 148
OUTREC..... 149

Token: 8F

Syntaxe:

```
OFF instruction
instruction OFF
```

L'instruction OFF annule l'effet de l'instruction ON.

OFF est utilisé avec:

- BREAK,
- EOF,
- ERR,
- ERRTOOL,
- EXCEPTION,
- KBD,
- GOTO,
- GOSUB,
- GRAF,
- TIMER.

Voir BREAK, EOF, ERR, ERRTOOL, EXCEPTION, KBD, GOTO, GOSUB, GRAF et TIMER.

Token: 90

Syntaxe:

```
ON instruction
instruction ON
```

L'instruction ON est utilisée pour valider chacune des instructions suivantes:

- BREAK,
- EOF,
- ERR,
- ERRTOOL,
- EXCEPTION,
- KBD,
- GOTO,
- GOSUB,
- GRAF,
- TIMER.

Voir BREAK, EOF, ERR, ERRTOOL, EXCEPTION, KBD, GOTO, GOSUB, GRAF et TIMER.

Token: A8

Syntaxe:

```
OPEN monfichier, FILTYP=type [FOR APPEND] AS £n
OPEN monfichier, FILTYP=type [FOR INPUT]  AS £n
OPEN monfichier, FILTYP=type [FOR OUTPUT] AS £n
OPEN nomfichier, FILTYP=type [FOR UPDATE] AS £n
```

La valeur de `n` sera comprise entre `0` et `29`. En mode différé mettre entre guillemets le nom du fichier à ouvrir.

Valeurs réservées:

```
0         .CONSOLE
30        EXEC
31        CAT, CATALOG et DIR
```

*ProDOS* 16 version 1.3 permet d'ouvrir 9 fichiers simultanément. **IIGS Basic** réserve un fichier pour `TYPE, CATALOG, INVOKE` et `LIBRARY`. Si, après avoir ouvert 8 fichiers, on tente d'en ouvrir un neuvième, le système envoie une erreur. Une version future de *ProDOS* permettra d'utiliser pleinement les possibilités du **IIGS Basic** (29 fichiers ouvert en même temps!).

Il est possible d'ouvrir un fichier texte pour y stocker un programme **IIGS Basic**. Pour ce faire la bonne solution est d'utiliser la ligne suivante:

```
OPEN monfichier, FILTYP=TXT AS £1 : OUTREC=0 : LIST : OUTPUT £0 : OUTREC=80 :
CLOSE £1
```

Utiliser:

- `FOR APPEND` pour écrire des données à la suite d'un fichier,
- `FOR INPUT` pour lire des données à partir d'un fichier,
- `FOR OUTPUT` pour écrire des données dans un fichier,
- `FOR UPDATE` pour créer un fichier puis y écrire des données.

Voir `OUTREC=, OUTPUT, FILTYP=, APPEND, INPUT, OUTPUT, UPDATE` et `CLOSE`.

Messages d'erreur possibles:

```
?DIRECTORY FULL ERROR
?FILE CREATE ERROR
?FILE NOT OPEN ERROR
?INT/FCB/VCB/ TBL FULL ERROR
```

Token: DF 89 Opérateur logique

Syntaxe: `PRINT a = b+c OR a = c-d`

L'opérateur logique `OR` teste la véracité des deux expressions entre lesquelles il se trouve et aboutit à une affirmation vraie (valeur `1`) si au moins l'une des deux est vraie. Si les deux expressions sont fausses, on a une affirmation fausse (valeur `0`).

Table de vérité:

| b \ a | 0 | 1 |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 1 | 1 |

Exemple:

```
PRINT 6 = 4 OR 12 <> 10
1
```

Token: 92

Syntaxe: `OUTPUT £n`

`OUTPUT £n` renvoie l'affichage écran vers le fichier spécifié par `n`. Les instructions `PRINT, LIST, TRACE, CAT, DIR` et `CATALOG` auront pour destination le fichier référencé `n` à son ouverture par `OPEN ... AS £n`.

Pour revenir à l'affichage écran, taper: `OUTPUT £0`.

Voir l'instruction `OPEN`.

Token: DF E6 Variable réservée

Syntaxe: OUTREC=n

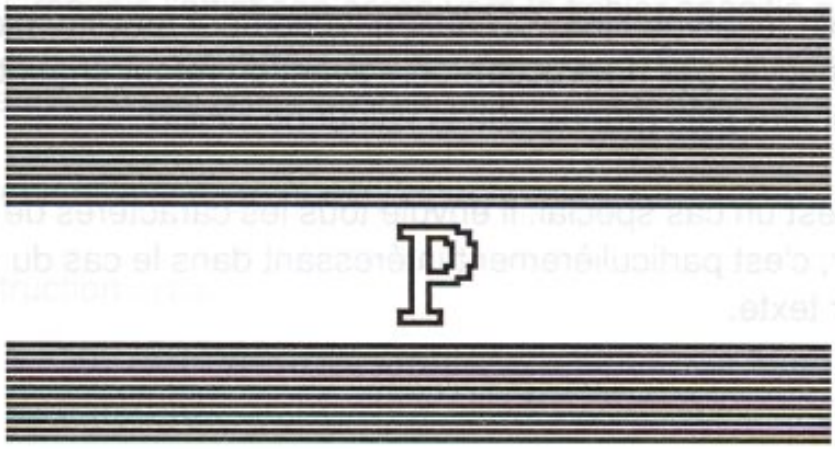
La variable réservée OUTREC= contient le nombre maximum de caractères à envoyer par l'instruction LIST avant un retour chariot. La valeur de n doit être plus grande que la valeur de INDENT.

OUTREC=0 est un cas spécial: il envoie tous les caractères de la ligne sans la couper; c'est particulièrement intéressant dans le cas du LIST, dans un fichier texte.

Exemple:

```
LOAD /GSB/DEMOS/PICS
OPEN PICS.TXT,FILTYP=TXT AS £1 : OUTREC=0 : OUTPUT £1 : LIST : OUTPUT £0:
OUTREC=80 : CLOSE £1
```

La première ligne charge le programme PICS. La deuxième ligne crée un fichier PICS.TXT du type TXT et liste le programme PICS. Pour visualiser la différence, supprimer OUTREC et exécuter à nouveau le OPEN avec PIC.TXT comme nom, après avoir enlevé les deux OUTREC= de la ligne, valider. Ensuite, taper: TYPE PICS.TXT et TYPE PIC.TXT




PDL()..... 151
PDL9..... 152
PEEK()..... 153
PERFORM..... 154
PFX$()..... 155
PI..... 156
POKE..... 157
POP..... 158
PREFIX..... 159
PREFIX$..... 160
PRINT..... 161
PROC..... 162
PROGNAM$..... 163
PUT..... 164

Token: DF 9C

Fonction

Syntaxe: `x = PDL(n)`

Quand on possède une paire de poignées de jeux, on peut utiliser la fonction `PDL()` pour lire leur position. La valeur de `n` est comprise entre `0` et `3` ce qui permet de lire la position de quatre poignées de jeux. `PDL()` retourne une valeur comprise entre `0` et `255`.

Exemple:

```
 10 REM PDL.GSB
 20 FOR I% = 0 TO 3
 30     PRINT "Valeur de PDL("I%"): "PDL(I%)
 40 NEXT
 50 END

RUN
Valeur de PDL(0): 0
Valeur de PDL(1): 0
Valeur de PDL(2): 0
Valeur de PDL(3): 0
```

Token: DF F3 Variable réservée

Syntaxe: `x = PDL9`

`PDL(n)` lit dans un registre 16 bits, effectue un `ASL` (multiplication par 2) de l'octet bas et stocke le résultat dans la variable réservée `PDL9`.

Exemple:

```
 10 REM PDL9.GSB
 20 FOR I% = 0 TO 3
 30     PRINT "Valeur de PDL("I%"): "PDL(I%)"  Valeur de PDL9: "PDL9
 40 NEXT
 50 END

RUN
Valeur de PDL(0): 0 Valeur de PDL9: 512
Valeur de PDL(1): 0 Valeur de PDL9: 512
Valeur de PDL(2): 0 Valeur de PDL9: 512
Valeur de PDL(3): 0 Valeur de PDL9: 512
```

L'exemple ci-dessus a été exécuté sans avoir de `JOYSTICK` connecté.

Token: DF B5 Fonction

Syntaxe: `x = PEEK(n)`

La fonction `PEEK()` lit l'octet de la mémoire spécifiée par `n`. La valeur de '`n`' sera comprise entre `0` et `2 ^ 24` (soit `1.677722E+07`), la valeur retournée sera comprise entre `0` et `255`.

Vous devez utiliser `PEEK()` avec précaution car lire en certaines cases mémoire peut causer des effets imprévus et imprévisibles.

Il est recommandé de ne pas utiliser cette fonction dans les programmes: ceux-ci seraient inutilisable sur de futures versions de l'*Apple IIGS* .

Token: B8

Syntaxe: `PERFORM nomfichier ([,var1] [,var2])`

`PERFORM` exécute une procédure en langage machine précédemment chargée par `INVOKE`.

Si une liste d'arguments est présente (entre parenthèses), chaque argument est évalué et passé à la procédure avant son exécution. Les arguments numériques sont convertis au format spécifié par la définition d'interface du dictionnaire de librairie.

Pour envoyer l'adresse d'une variable numérique, utiliser la fonction `VARPTR()`. On peut également passer l'adresse d'une chaîne de caractères en utilisant la fonction `VARPTR$()`.

Messages d'erreur possible:

```
?ARGUMENT TYPE MISMATCH
?ARGUMENT COUNT ERROR
```

Token: DF BD Fonction

Syntaxe: `PRINT PFX$(n)`

La variable réservée `PFX$()` retourne la chaîne de caractères correspondant au préfixe `n`. La valeur de `n` est comprise entre `0` et `8`. `PFX$(8)` correspond toujours au préfixe de démarrage. Les autres préfixes sont à définir. Le **IIGS Basic** alloue les préfixes suivant:

```
PFX$(0) = /GSB/
PFX$(1) = /GSB/
PFX$(2) = /GSB/SYSTEM/LIBS
PFX$(3) =
PFX$(4) =
PFX$(5) =
PFX$(6) =
PFX$(7) =
PFX$(8) = /GSB
```

On peut récupérer le préfixe dans une variable de chaîne: `A$ = PFX$(2)`.

Exemple:

```
FOR A = 0 TO 8 : PRINT PFX$(A) : NEXT

/GSB/
/GSB/
/GSB/SYSTEM/LIBS



/GSB
/GSB
```

Token: DF F4 Variable réservée

Syntaxe: `x = PI`

La variable réservée `PI` contient la valeur exacte de `PI` sur un nombre de `20` chiffres.

`PI` peut être converti en simple ou double précision dans une variable.

L'affichage de `PI` sera fonction de la valeur de `SHOWDIGITS`. Voir donc cette instruction.

Exemple:

```
 5 REM   PI.GSB
10 TEXT:HOME:VPOS=10
20 PRINT "Démonstration de l'instruction PI":PRINT
30 INPUT "Rayon du cercle:  ";R
40 PRINT "Périmètre:        ";2*PI*R
50 VPOS=20:END
```

Token: D2 Fonction

Syntaxe: `POKE adr,val`

La fonction `POKE` écrit la valeur `val` dans l'octet de la mémoire spécifiée par `adr`. La valeur de `adr` sera comprise entre `0` et `22^4` (soit `1.677722E+07`), la valeur de `val` sera comprise entre `0` et `255`.

Utiliser `POKE` avec précaution car écrire dans certaines cases mémoire peut causer des effets imprévus.

Il est recommandé de ne pas utiliser cette fonction dans les programmes: ceux-ci ne fonctionneraient pas dans de futures versions de l'*Apple IIGS*.

Token: BB

Syntaxe: POP

L'instruction POP permet de sauter un RETURN et donc de revenir à l'instruction située après le GOSUB qui suit celui qui a conduit à ce sous-programme.

Exemple:

```
 10        REM POP.GSB
 20 DEBUT:GOSUB 50
 30        PRINT "Ici la ligne 30"
 40 FIN:   END
 50        GOSUB 80
 60        PRINT "Ici la ligne 60"
 70        END
 80        POP
 90        RETURN
100        END

TRACE
RUN
  £10  £20  £50  £80  £90  £30 Ici la ligne 30
  £40
```

Token: 9B

Syntaxe:

**1 -** `PREFIX`
**2 -** `PREFIX ?`
**3 -** `PREFIX n`
**4 -** `PREFIX /nomvolume ou nomsouscatalogue ou n`
**5 -** `PREFIX n,/nomvolume ou nomsouscatalogue ou n`

**1 -** `PREFIX` écrit la chaîne contenue dans `PREFIX 0`,
**2 -** Tous les préfixes sont affichés,
**3 -** Affiche le préfixe `n`, la valeur de `n` est comprise entre `0` et `8`
**4 -** On assigne au préfixe `0` un nom de disquette ou de sous-catalogue.
**5 -** On assigne au préfixe `n` un nom de disquette, de sous-catalogue.

En mode différé, il faut mettre entre guillemets la chaîne de caractères `nomvolume` ou `nomsouscatalogue`.

Exemple:

```
PREFIX ?
0 /GSB/
1 /GSB/
2 /GSB/SYSTEM/LIBS/
3
4
5
6
7 /GSB/
```

Token: DF E4 Variable réservée

Syntaxe: `A$ = PREFIX$`

`PREFIX$` est une variable réservée qui contient le plus récent préfixe. Le contenu de `PREFIX$` est identique à celui de `PREFIX 0` et de `PFX$(0)`

Exemple:

```
PRINT PREFIX$

/IIGS.BASIC/
```

Token: CE

Syntaxe:

**1 -** `PRINT [TAB(n)] [SPC(n)] ["carac"] [;var] [,var$] [,var%]`
**2 -** `PRINT USING nl; ["carac"] [;var] [,var$] [,var%]`
**3 -** `PRINT [£n] [,recnum] ; ["carac"] [;var] [,var$] [,var%]`
**4 -** `PRINT [£n] [,recnum] USING nl; ["carac"] [;var] [,var$] [,var%]`

**1 -** `PRINT` affiche du texte sur l'écran. Ses arguments optionnels sont:

- une chaîne de caractères entre guillemets,
- une variable de chaîne,
- une variable numérique,
- la fonction `TAB()`,
- la fonction `SPC()`,
- la fonction `SPACE$()`,

séparés par un point virgule (;) ou par une virgule (,).

**2 -** `PRINT £n` affiche du texte sur la sortie précisée par `£n`.

**3 -** `PRINT USING nl` affiche du texte en utilisant le formatage précisé par l'instruction `IMAGE` de la ligne `nl`.

**4 -** `PRINT £n USING nl` affiche du texte sur le périphérique `£n` en utilisant le formatage précisé par l'instruction `IMAGE` de la ligne `nl`.

Voir les fonctions `TAB()`, `SPACE$()` et `SPC()`, `OPEN`, `CREATE`, `IMAGE` ainsi que la variable réservée `SHOWDIGITS`.

Exemple:

```
10 REM PRINT.GSB
20 CREATE "PRINT.TXT", FILTYP=TXT
30 OPEN "PRINT.TXT", AS £1
40 PRINT£1 "Démonstration de PRINT £n"
50 PRINT£1 "Vous utiliserez cette instruction pour créer un fichier EXEC"
60 PRINT£1 "Vous pouvez également stocker des informations dans un fichier"
70 PRINT£1 "texte, puis les relire avec TYPE ou les reprendre dans un"
80 PRINT£1 "programme à l'aide de INPUT£n."
90 CLOSE£1
100 END
```

Token: 88 Fonction

Syntaxe: `DEF PROC nomproc(arg1 [,arg2])`

La fonction PROC doit être associée à la fonction DEF pour définir une procédure. Voir l'instruction DEF pour la définition d'une procédure. Ensuite pour appeler la procédure, il faut taper PROC, suivi du nom de la procédure et de son ou ses arguments.

Exemple:

```
  5 REM  PROC.GSB
 10 DEF PROC centre(a$)
 20 LOCAL a%
 40 a%=40-LEN(a$) DIV 2
 50 HPOS=a%:PRINT a$
 60 END PROC centre
 70 PROC centre("Démonstration")
 80 PROC centre("d'une procédure")
 90 PROC centre("d'affichage d'un texte")
100 PROC centre("centré")
110 PRINT
120 PROC centre("......Ce message fait 78 caractères afin de vérifier le
centrage.............")
130 PRINT
140 PROC centre("Cette procédure ne peut pas traiter les messages")
150 PROC centre("d'une grandeur supérieure à 78 caractères")
160 PROC centre("A vous d'essayer de faire mieux!")
170 END

RUN
                              Démonstration
                            d'une procédure
                         d'affichage d'un texte
                               centré

......Ce message fait 78 caractères afin de vérifier le centrage.....

                  Cette procédure ne peut pas traiter les messages
                    d'une grandeur supérieure à 78 caractères
                         A vous d'essayer de faire mieux!
```

Token: DF E5 Variable réservée

Syntaxe: `PRINT PROGNAM$`

La variable réservée `PROGNAM$` contient le nom du dernier programme chargé.

Exemple:

```
PRINT PROGNAM$
PROC.GSB

PRINT PREFIX$;PROGNAM$
/GSB/PROG.GSB
```

Token: D5

Syntaxe:

```
PUT £n [,longueur] [,numreg] ; varchaine
```

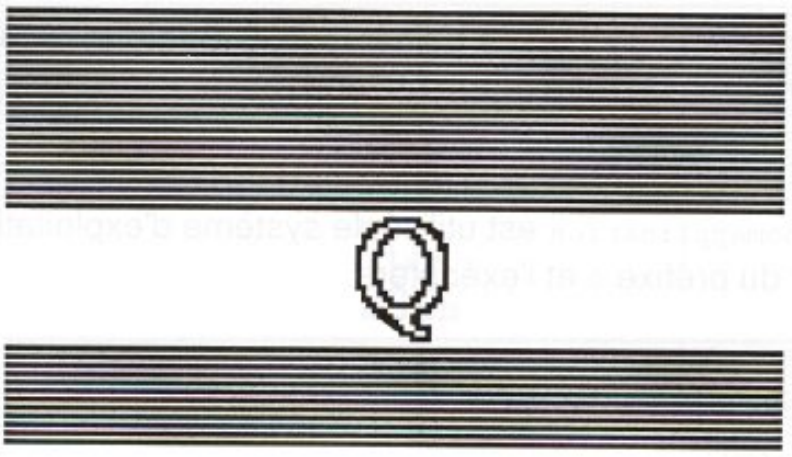
L'instruction PUT transfère x octets d'un tableau d'octets vers un fichier à accès séquentiel £n, dans l'enregistrement numreg. La valeur de x est égale à la longueur de l'enregistrement sauf si un paramètre longueur est précisé.

Quand le numreg est de 1, le paramètre longueur est utilisé. Quand l'option numreg n'est pas utilisée, le transfert débute à la position actuelle, mais la grandeur du transfert est limitée à la fraction restante de l'enregistrement actuel. GET£n et PUT£n peuvent manipuler tous les types de fichiers.

L'instruction PUT£n limite la taille d'un transfert à la dimension la plus à gauche d'une structure d'octets ou à la taille maximum d'un enregistrement soit 32767 octets.

Exemple:

```
  5       REM  PUT.GSB
 10 DEBUT:DIM pic!(32767)
 20       OPEN "/gsb/demos/pic1", FILTYP=6 FOR INPUT AS £1
 30       GET£1,32767,0;pic!(0)
 40       CLOSE£1
 50       PRINT "Le contenu de l'image pic1 est maintenant dans le tableau
          pic!(0)"
 60       OPEN "/gsb/demos/pic1.bis", FILTYP=6 FOR OUTPUT AS £1
 70       PUT£1,32767,0;pic!(0)
 80       CLOSE£1
 90       PRINT "Le contenu du tableau pic!(0) a été sauvé dans le fichier
          pic1.bis"
100 FIN: END
```




QUIT..... 166

Token: A9 Commande

Syntaxe: `QUIT [nomapplication]`

La commande `QUIT` permet de quitter le **IIGS Basic** et redonne le contrôle au programme de démarrage ou bien au sélecteur *Apple IIGS Start Next Program* .

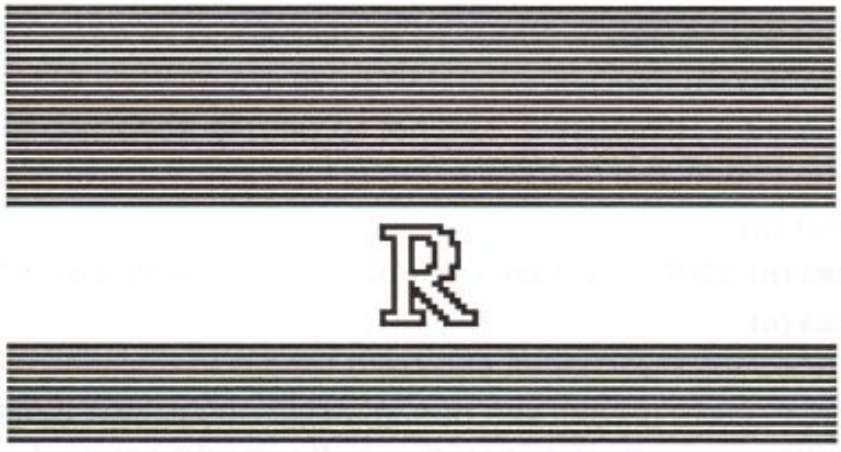
Si l'option `nomapplication` est utilisé, le système d'exploitation le chargera à partir du préfixe `0` et l'exécutera.

Exemple:

```
PREFIX /SYSTEM.DISK/
QUIT BASIC.SYSTEM
```

On donne `/SYSTEM.DISK/` comme préfixe `0`, et on demande à quitter le **IIGS Basic** pour aller au `BASIC.SYSTEM`. Le système charge donc `ProDOS 8`, `BASIC.SYSTEM` et les exécute tous les deux.

```
 5 REM QUIT.GSB
10 PRINT "Démonstration de l'instruction QUIT /nomvolume/application"
20 PRINT:PRINT "Attention, répondre 'Oui' ci-dessous vous fera exécuter
l'application choisie!"
30 INPUT "Voulez-vous quitter le GS Basic? (O/N) ";x$
40 IF x$="O" OR x$="o" THEN quitte
50 TEXT:HOME:NEW:END
60 QUITTE:INPUT "Nom ProDOS COMPLET de l'application choisie: ";suite$
70        QUIT suite$
```




R.STACK%(), R.STACKà(), R.STACK&() ..... 168
RANDOMIZE ..... 169
READ ..... 170
REC() ..... 171
RELATION() ..... 172
REM ..... 173
REMDR ..... 174
RENAME ..... 175
RENUM ..... 176
REP$() ..... 177
RESTORE ..... 178
RESUME ..... 179
RETURN ..... 180
RIGHT$() ..... 181
RND() ..... 182
ROUND() ..... 183
RUN ..... 184

Token:

1 - R.STACK%() DF 9E
2 - R.STACK&() DF A0
3 - R.STACKà() DF 9F

Syntaxe:

```
1 - x = R.STACK%(n)
2 - x = R.STACK&(n)
3 - x = R.STACKà(n)
```

La fonction R.STACK() retourne des informations prises dans la pile de retour de la fonction CALL; c'est-à-dire d'un buffer de 32 octets. Les fonctions CALL et CALL% retournent 32 octets ou 16 mots dans l'exécution des fonctions de la boîte à outils, dans une zone mémoire réservée.

L'octet non-signé n est un pointeur dans la pile de retour d'informations. Il aura une valeur comprise entre 0 et 16 pour R.STACK%(), 0 et 15 pour R.STACKà(), et entre 0 et 13 pour R.STACK&().

R.STACKà() retournera un double mot soit 32 bits (4 octets) dans une variable entière signée. R.STACK%() retournera un entier soit 16 bits (2 octets) dans une variable entière signée. Enfin R.STACK&() retournera un entier long soit 64 bits (8 octets) dans une variable entière signée.

R.STACK%(0) retournera un mot contenant le nombre de mots perdus dans la pile d'informations à la suite de la dernière exécution de CALL.

R.STACK%(1) retournera le mot qui était en haut de la pile d'informations après le retour de l'exécution d'une fonction de la boîte à outils.

La pile d'informations sera effacée lors de chaque exécution de la fonction CALL ou CALL%. On doit devez donc récupérer les données entre chaque utilisation de CALL ou de CALL%

Token: D0 Instruction

Syntaxe: `RANDOMIZE n`

La fonction `RANDOMIZE` initialise le générateur de nombres aléatoires. L'argument `n` est le nombre qui est utilisé comme point de départ. Il doit avoir une valeur comprise entre `1` et `21^6383`. L'expression peut venir de la variable réservée `SECONDSà` après l'exécution d'une instruction `TIMER ON`.

La variable réservée `SECONDSà` a une valeur comprise entre `0` et `86399`.

Si le générateur de nombres n'est pas initialisé, la fonction `RND()` renvoie la même suite de nombres.

Token: AC

Syntaxe:

**1 -** `READ var1 [,var2]`
**2 -** `READ £numfichier [,numenregistrement] [;var1] [,var2]`

**1 -** `READ`

L'instruction READ lit les valeurs définies par l'instruction DATA et les affecte à des variables.

Si le type de variable n'est pas en accord avec le type de valeur on aboutit à une erreur. Si le nombre de variables suivant READ dépasse le nombre des valeurs contenues dans la ligne de DATA on a aussi une erreur.

**2 -** `READ £n`

L'instruction READ £ lit les informations à partir d'un fichier de type BDF spécifié par le numéro de fichier. Un numéro d'enregistrement optionnel peut lui être spécifié pour lire les fichiers à accès séquentiels.

Les variables qui suivent READ £ doivent être de même type que les informations lues dans le fichier.

Messages d'erreur possibles:

```
?SYNTAX ERROR
?OUT of DATA
```

Exemple:

```
 10         REM READ.GSB
 20 DEBUT:  Fichier$="READ.BDF"
 30         OPEN Fichier$, FILTYP= BDF FOR INPUT AS£1
 40 BOUCLE:BREAK ON
 50         FOR A%=1 TO 50
 60             ON EOF£1 GOTO 100
 70             READ£1;Ligne$,Num
 80             PRINT Ligne$,Num;" ";
 90         NEXT
100         CLOSE£1
110 FIN:    END

RUN
Ligne numéro 1 Ligne numéro 2 Ligne numéro 3 Ligne numéro 4 Ligne numéro 5
Ligne numéro 6 Ligne numéro 7 Ligne numéro 8 Ligne numéro 9 Ligne numéro 10
```

Token: DF 9A

Fonction

Syntaxe: `x = REC(numfichier)`

La fonction `REC()` retourne le numéro d'enregistrement du fichier spécifié par `numfichier`.

Si on utilise l'instruction `INPUT£` ou `READ£` pour lire le catalogue d'un volume ou d'un sous-catalogue, `REC()` retourne le numéro de la ligne actuellement lue.

Token: DF C1 Fonction

Syntaxe:

```
PRINT RELATION(var1,var2)
a = RELATION(var1,var2)
```

La fonction RELATION() renvoie une valeur en fonction de la relation existante entre ses deux arguments.

| Valeur retournée | Valeur argument var1 | Valeur argument var2 | Signification |
|---|---|---|---|
| 1 | 2 | 1 | var1 est plus grand que var2 |
| 2 | 1 | 2 | var1 est plus petit que var2 |
| 3 | 1 | 1 | var1 est égal à var2 |

RELATION() possède deux arguments qui peuvent être une variable numérique ou un nombre. Il n'est pas permis d'utiliser une chaine de caractère ou une variable de chaine.

Exemple:

```
 10         REM RELATION.GSB
 20 DEBUT:  B=20:REM Valeur fixe
 30 BOUCLE:FOR A=10 TO 30
 40             ON RELATION(A,B) GOSUB 100,120,140
 50         NEXT
 60 FIN:    END
100 GRAND:  PRINT "A est plus grand que B. A = "A" B = "B
110         RETURN
120 PETIT:  PRINT "A est plus petit que B. A = "A" B = "B
130         RETURN
140 EGAL:   INVERSE:PRINT "A est égal à B.          A = "A" B = "B;
150         NORMAL:PRINT
160         RETURN
```

Token: C8 Mot réservé

Syntaxe: `REM commentaire`

Le mot réservé REM (pour remarque) permet d'insérer des commentaires dans un programme. On ne peut utiliser plus de 239 caractères après l'instruction REM.

Utiliser plusieurs lignes de REM plutôt qu'une seule très longue. Ne pas oublier que l'abondance de REM augmente l'encombrement mémoire et ralentit l'exécution du programme.

Token: DF 86 Fonction

Syntaxe:

```
PRINT 12 REMDR 4
X = 12.482 REMDR 4.824
```

L'instruction REMDR (Remainder, en français reste) a été incluse dans le **IIGS Basic** pour être utilisée avec *SANE*.

REMDR retourne le plus petit reste possible contrairement à MOD.

Se reporter au manuel *Apple arithmetics - SANE -*

Exemple:

```
PRINT 12.123456 REMDR 5.98765
0.1481562

PRINT 12.123456 MOD 5.98765
0
```

Token: A1 Commande

Syntaxe: `RENAME nomfichier1,nomfichier2 [,FILTYP=n]`

L'instruction `RENAME` permet de changer le nom d'un volume, d'un sous-catalogue ou d'un fichier.

La liste des arguments de `RENAME` comprend:

1 - l'ancien nom,
2 - une virgule séparatrice,
3 - le nouveau nom,
4 - l'option `FILTYP=n` qui permet d'attribuer au fichier le type `n` et a pour valeur un nombre compris entre `0` et `255` ou bien le code mnémonique du type désiré.

On peut seulement changer le type de fichier en donnant le même nom aux arguments 1 et 3, puis en 4 donner le nouveau type de fichier.

Exemple:

```
RENAME /SYSTEM.DISK/STARTUP,/SYSTEM.DISK/HELLO
```

Token: 85

Syntaxe: `RENUM [lignedebut] [,incrément] [,numligne1 - numligne2]]`

La commande RENUM permet de renuméroter les lignes du programme actuellement en mémoire.

La valeur par défaut de la ligne de départ et de l'incrément est 10. Les deux derniers paramètres optionnels, sont à utiliser pour renuméroter ou déplacer une portion de programme.

RENUM remplace les anciens numéros de ligne qui suivent les ON ... GOSUB ou ON ... GOTO et autres IF ... THEN et IF ... GOTO par la nouvelle valeur.

Avant d'utiliser RENUM, sauver le programme. En effet, la version 1.0B4 du **IIGS Basic** associée à la **ROM 0** est très peu fiable en ce qui concerne le RENUM, et il perd très facilement le programme ou "plante" le système.

Token: DF D3 Fonction

Syntaxe: `x$ = REP$("carac",n)`

La fonction `REP$()` retourne `n` fois le premier caractère de la chaîne ou variable de chaîne `carac`. La valeur de 'n' doit être comprise entre `0` et `255`.

Exemple:

```
PRINT REP$("JIBENA",5)
JJJJJ
```

Token: C5

Syntaxe:

```
RESTORE [numligne]
RESTORE [label]
```

L'instruction `RESTORE` déplace le pointeur de lecture des datas au début de la liste de data de façon à permettre de relire la liste. L'argument optionnel `numligne` ou `label` autorise une relecture sélective de la ligne désirée.

Si la ligne donnée en argument n'est pas une ligne de data, une erreur apparaît.

Message d'erreur possible:

```
?INVALID LINE NUMBER/LABEL
```

Token: C2

Syntaxe:

**1 -** RESUME
**2 -** RESUME [NEXT]
**3 -** RESUME [COPY]

**1 -** RESUME

L'instruction RESUME réexécute l'instruction responsable d'une erreur après correction par un sous-programme. RESUME sans option n'a d'effet que s'il est utilisé avec une routine ON ERR.

**2 -** RESUME NEXT

RESUME NEXT doit être utilisé à la place de RESUME pour exécuter l'instruction qui suit celle responsable de l'erreur.

**3 -** RESUME COPY

RESUME COPY doit être utilisé pour rendre le contrôle à l'instruction COPY pour effectuer une copie avec un seul lecteur de disquettes.

Token: C7

Syntaxe: `RETURN [0]`

L'instruction `RETURN` permet de revenir au programme principal. Si le programme rencontre un `RETURN` de plus qu'il n'y a de `GOSUB`, une erreur apparaît.

Pour 'sauter' un `RETURN` utiliser `POP : GOTO numligne` ou `POP : GOTO label`.

`RETURN 0` est un cas spécial de `RETURN` qui est utilisé à la fin d'une routine de traitement des événements définie par `EVENTDEF` et `MENUDEF` pour être utilisée avec `TASKPOLL`.

Message d'erreur possible:

```
?RETURN w/o GOSUB
```

Token: DF D2

Fonction

Syntaxe: `A$ = RIGHT$(B$,n)`

La fonction `RIGHT$()` fournit les `n` caractères de droite de la chaîne `B$`.

Si `n` a une valeur supérieure à la longueur de la chaîne `B$`, la fonction donnera `B$`. Si `n` est égal à 0, une chaîne nulle sera renvoyée.

Exemple:

```
PRINT RIGHT$("Le petit chat noir miaule",6)
miaule
```

Token: DF A2 Fonction

Syntaxe: `x = RND(n)`

La fonction `RND()` renvoie un nombre aléatoire inférieur à 1. Un nouveau nombre sera renvoyé aussi souvent que l'argument `n` aura une valeur plus grande que zéro. Initialiser `RANDOMIZE` de façon à générer des nombres vraiment aléatoires.

Exemple:

```
10 RANDOMIZE 1
20 FOR A% = 0 TO 10
30     PRINT RND(1)
40 NEXT
50 END

RUN
0.7082972
```

Token: DF B6 Fonction

Syntaxe: `PRINT ROUND(n)`

La fonction `ROUND()` retourne la valeur intégrale arrondie de `n`, en accord avec l'arrondi de *SANE*. `ROUND` doit être utilisé à la place de `INT` car `INT` ne respecte pas le protocole *SANE*.

Token: A0 Commande

Syntaxe:

```
RUN [nomfichier]
RUN [numligne]
```

La commande RUN est utilisée pour lancer un programme. Avec RUN, le **IIGS Basic** ferme tout fichier ouvert sauf un fichier EXEC et efface toutes les variables.


SAVE..... 186
SCALB()..... 187
SCALE()..... 188
SECONDSà..... 189
SET..... 190
SGN()..... 191
SHOWDIGITS..... 192
SIN()..... 193
SPACE$()..... 194
SPC()..... 195
SQR()..... 196
SRC..... 197
STEP..... 198
STOP..... 199
STR$()..... 200
SUB$()..... 201
SWAP..... 202

Token: 9E Commande

Syntaxe:

```
SAVE [nomfichier]
SAVE AS nomfichier
```

La commande SAVE fait une copie sur disque du programme **IIGS Basic** actuellement en mémoire. nomfichier est optionnel car **IIGS Basic** conserve dans la variable réservée PROGNAM$ le nom du dernier programme chargé.

Pour changer le nom du programme actuel utiliser: SAVE AS nomfichier.

Attention: employer SAVE sans nom de fichier efface le fichier de ce nom et le remplace par le programme actuellement en mémoire; **AUCUN** contrôle n'est effectué lors de l'opération d'écriture, sauf la vérification du type du fichier.

Token: DF C4 Fonction

Syntaxe: `PRINT SCALB(n1;n2)`

`SCALB()` élève l'expression `n2` à la puissance $2^{n1}$. Soit la formule suivante: `n2 * 2 ^ n1`.

Voir l'instruction `LOGB%()`.

Exemple:

```
PRINT "SCALB(5,10): "SCALB(5,10)"  soit: 10 * 2 ^ 5: "10 * 2 ^ 5
SCALB(5,10): 320  soit: 10 * 2 ^ 5: 320
```

Token: DF C5 Fonction

Syntaxe: `PRINT USING "$$7£.2£";SCALE(n1,n2)`

`SCALE()` est utilisé conjointement à `PRINT USING` pour déplacer le point décimal vers la droite ou la gauche, en tronquant le nombre de la valeur `n1`. `SCALE` utilise deux arguments: `n1` détermine la position du point décimal dans le nombre, à compter de la droite, et l'argument `n2` qui est la valeur numérique à afficher.

Exemple:

```
A& = 1234567890 : PRINT USING "$7£.2£";SCALE(-4,A&)
$123,456.79
```

Token: DF EC

Variable réservée

Syntaxe: `A = SECONDSà`

`SECONDSà` est une variable réservée qui retourne la valeur du compteur initialisé par `TIMER ON`.

Si `TIMER ON` n'a pas été initialisé, `SECONDSà` retourne `0`. Après avoir exécuté `TIMER OFF`, le nombre retourné a une valeur inchangée. La valeur retournée sera comprise entre -1 et 86400. En effet il y a 86400 secondes dans une journée de 24 heures...

De part la conception de l'*Apple IIGS* , la valeur retournée par `SECONDSà` ne peut pas être exacte; en effet, un accès disque par exemple provoquera une interruption de l'incrémentation du compteur.

Exemple:

```
10          TIMER ON
15 BOUCLE:  BREAK ON
20          FOR A% = 0 TO 32766
30          NEXT
40          PRINT SECONDSà
50          TIMER OFF
60          BREAK OFF
70          END

RUN
60576
```

Token: D6

Syntaxe:

```
1 - SET (svar!(pos) [,n]) = val
2 - SET (svar!(pos) [,n]) = [^] a$
3 - SET (svar!(pos) [,n]) = *adrmem [,noctets]
```

L'instruction SET permet de stocker une variable ou une expression dans une structure.

1 - Stocke la valeur `val` dans le tableau `svar!()` à partir de la case `pos` sur `n` octets de long.

| Type de variable | Suffixe: | Longueur |
|---|---|---|
| réelle en simple précision | | 4 octets |
| réelle en double précision | £ | 8 octets |
| simple entière | % | 2 octets |
| double entière | à | 4 octets |
| entière longue | & | 8 octets |
| chaîne | $ | long. chaine (0...255 octets) |

2 - Stocke la chaîne `a$` dans le tableau `svar!()` à partir de la case `pos` sur `n` octets de long. Si vous utilisez l'option `^`, la chaîne sera stockée avec l'octet longueur en début dans le tableau `svar!` à partir de la case `pos` sur `n` octets de long.

3 - Stocke le contenu de l'emplacement mémoire `adrmem` sur `noctets` de long (si spécifié) dans le tableau `svar!()` à partir de la case `pos` sur `n` octets de long.

Token: DF 96 Fonction

Syntaxe: `x = SGN(n)`

La fonction `SGN()` retourne le signe de `n`.

Si `n` est négatif, `SGN(n)` donne `-1`
Si `n` est positif, `SGN(n)` donne `1`
Si `n` est égal à zéro, `SGN(n)` donne `0`

Exemple:

```
PRINT SGN(-4),SGN(0),SGN(4)
-1        0          1
```

Token: DF E8 Variable réservée

Syntaxe:

```
SHOWDIGITS = n
LET SHOWDIGITS = n
```

La valeur par défaut de la variable réservée SHOWDIGITS est 7. Sa valeur maximum est 28. SHOWDIGITS contrôle le nombre de chiffres significatifs qui seront imprimés par PRINT, sans pour cela influencer PRINT USING.

PRINT formate les variables en notation scientifique si la valeur de SHOWDIGITS est plus petite que le nombre de chiffres à afficher.

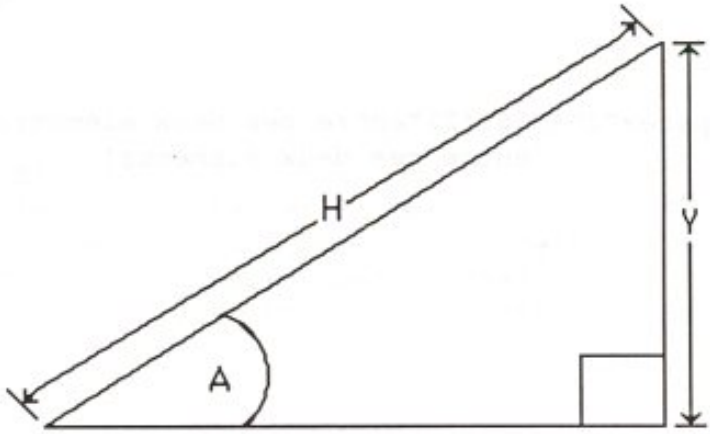
Exemple:

```
PRINT PI
3.141593

SHOWDIGITS=28:  PRINT PI
3.1415926535897932385128089 59
```

Token: DF AB Fonction

Syntaxe: `n = SIN(A)`

`SIN()` retourne le sinus d'un angle `A` exprimé en radians. L'argument `n` représente la valeur de l'angle `A`.



Token: DF BE
Fonction

Syntaxe: `PRINT SPACE$(n)`

`SPACE$()` retourne une chaîne de `n` espaces. La valeur de `n` est comprise entre `0` et `255`.

Exemple:

```
PRINT "Il y a 12 espaces"SPACE$(12)"entre ces deux éléments!"
Il y a 12 espaces            entre ces deux éléments!
```

Token: DF 82 Fonction

Syntaxe: `PRINT SPC(n)`

La fonction `SPC()` permet d'ajouter `n` espaces à l'affichage dans une instruction `PRINT`, à partir de la position actuelle du curseur.

Voir `PRINT`.

Exemple:

```
10        REM  SPC.GSB
20 DEBUT:HOME:PRINT "Démonstration de la fonction SPC()"
30        VPOS=10:PRINT "Faire attention à cette caractéristique de SPC()"
40        PRINT "Pressez une touche";:GET$A$
50        VPOS=10:HPOS=18:PRINT SPC(6)"ce 'bug ?'" SPC(6)
60 FIN:   END
```

Token: DF A1 Fonction

Syntaxe: `x = SQR(n)`

`SQR()` retourne la racine carrée de la valeur `n`.

Exemple:

```
PRINT SQR(25)
5
```

Token: DF 8C

Type de fichier texte utilisé par *APW* et par le **IIGS Basic**.

Utiliser SRC à la création ou à l'ouverture d'un fichier.

Un fichier SRC créé à partir du **IIGS Basic** a pour sous-type $BA (186).

Exemple:

```
OPEN monfichier, FILTYP= SRC AS £1
```

ou bien:

```
CREATE monfichier, FILTYP= SRC
```

Token: DF 87

Syntaxe: `FOR a = b TO c STEP d : NEXT`

`STEP` permet d'incrémenter ou de décrémenter d'une valeur entière `d` la variable de contrôle d'une boucle `FOR ... NEXT`. Sa valeur par défaut est `1`.

Voir les instructions `FOR` et `NEXT`.

Token: C9

Syntaxe: STOP

STOP arrête l'exécution du programme et provoque le retour en mode direct.

STOP affiche le message:

```
?PROGRAM INTERRUPTED nnnn
```

où nnnn sera le numéro de la ligne d'interrruption.

Token: DF BB Fonction

Syntaxe: `a$ = STR$(n)`

`STR$()` évalue la valeur de l'expression `n` et retourne la valeur sous forme d'une variable de chaîne.

Voir les instructions `SUB$(), RIGHT$(), LEFT$(), TEN(), MID$(), HEX$(), VAL(), UCASE$()`.

Exemple:

```
A$ =  STR$(12) : PRINT A$
12
```

Token: BD Fonction

Syntaxe: `a$ = "chaine" : SUB$(a$,n) = "C" : PRINT a$`

`SUB$` permet de remplacer une portion de chaîne définie par la valeur `n` ( `n` = position de la chaîne de remplacement), par une autre chaîne.

Exemple:

```
a$ = "chaine" : SUB$(a$,1) = "C" : PRINT a$
Chaine

a$ = "essai 1" : SUB$(a$,7) = "2" : PRINT a$
essai 2
```

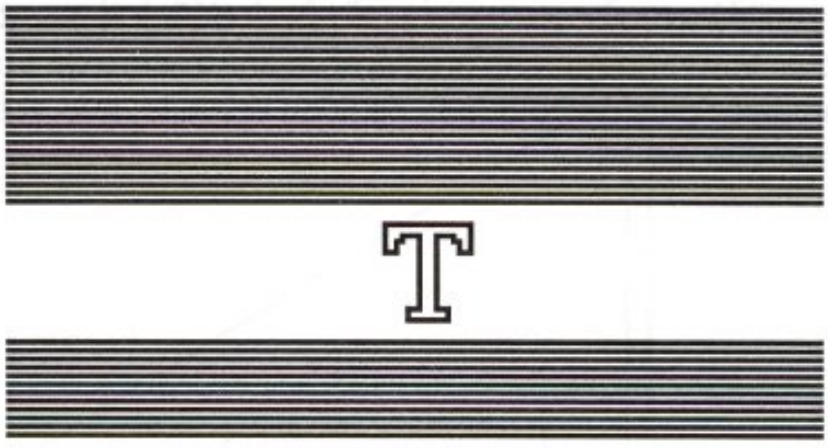
Token: C6 Mot réservé

Syntaxe: `SWAP var1,var2`

`SWAP` permute la valeur des deux variables. On peut utiliser les variables de chaîne, entières et réelles avec `SWAP`, mais les deux variables doivent être du même type.

Exemple:

```
 5 REM   SWAP.GSB
10 A = 10 : B = 20
20 PRINT "A = "A, "B = "B
30 SWAP A,B
40 PRINT "Nouvelles valeurs:"
50 PRINT "A = "A, "B = "B
60 END

RUN

A = 10         B = 20
Nouvelles valeurs:
A = 20         B = 10
```




TAB() ..... 204
TAN() ..... 205
TASKPOLL ..... 206
TASKREC%() ..... 207
TEN() ..... 208
TEXT ..... 209
TEXTPORT ..... 210
THEN ..... 211
TIME$ ..... 212
TIME() ..... 213
TIMER ..... 214
TO ..... 215
TRACE ..... 216
TXT ..... 217
TYP() ..... 218
TYPE ..... 219

Token: DF 80

Syntaxe: `PRINT TAB(n)`

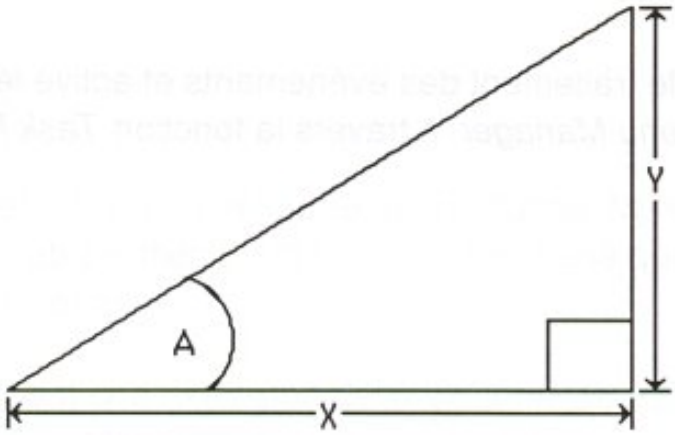
`TAB()` est utilisé avec l'instruction `PRINT` pour définir le nombre d'espaces depuis la marge gauche à insérer avant d'écrire du texte. Si on donne à `n` une valeur plus petite que la position actuelle, aucun espace n'est inséré devant le caractère suivant.

Token: DF AC

Fonction

Syntaxe: `n = TAN(A)`

`TAN()` retourne la tangente de l'angle `A` exprimé en radians.




La tangente `TAN()` est définie comme le rapport entre la longueur du côté opposé à l'angle et celle du côté contigu: `TAN(A) = Y/X`.

Token: AF

Syntaxe:

```
TASKPOLL INIT n1,n2
TASKPOLL ON
TASKPOLL OFF
```

TASKPOLL initialise le traitement des événements et active les *Window Manager* et *Menu Manager* à travers la fonction *Task Master* du *Window Manager* .

Token: Fonction

**1 -** `TASKREC%(` `DF B7`
**2 -** `TASKRECà(` `DF B8`

Syntaxe:

**1 -** `TASKREC%(n)`
**2 -** `TASKRECà(n)`

La fonction `TASKREC()` retourne le résultat du *Task Master TaskRec* en un simple ou double entier. `n` est un octet non signé qui a une valeur comprise entre `0` et `255`.

`TASKREC()` est une extension de la stucture d'enregistrement des taches du *Window Manager*. Le premier mot de `TASKREC()` est le masque de l'*Event Manager* utilisé par le **IIGS Basic** quand il appelle le *Task Master*. Pour plus de précision, se reporter au manuel de la boîte à outils, dans le chapitre *Window Manager* .

Token: DF CA

Fonction

Syntaxe: `PRINT TEN("$n")`

`TEN` retourne en décimal (base 10) son argument `n` donné en hexadécimal. La valeur retournée tient dans une variable entière double. L'expression `n` peut contenir en en-tête des espaces et un signe dollar (`$`) mais ne doit pas dépasser les huits chiffres, sous peine d'erreur.

Voir la fonction `HEX$`.

Message d'erreur possible:

```
?ILLEGAL QUANTITY ERROR
```

Exemple:

```
PRINT TEN("FFFF")
65535
```

Token: 93

Syntaxe: TEXT

L'instruction TEXT permet de revenir à l'écran texte standard lorsqu'on se trouve avec TEXTPORT ou GRAF ON actifs.

Token: B7

Syntaxe: `TEXTPORT hng,vnh TO hnd,vnb`

`TEXTPORT` vous de définir une fenêtre d'affichage.

Les coordonnées du rectangle s'affichent comme suit:

`hng` valeur `n` du point horizontal haut gauche,
`vnh` valeur `n` du point vertical haut,

`hnd` valeur `n` du point horizontal bas droit,
`vnb` valeur `n` du point vertical bas.

Chaque argument est une expression arithmétique positive et ne dépassant pas `255`, sinon on génère une erreur: `?ILLEGAL QUANTITY ERROR`.Si on utilise des valeurs dépassant la taille de l'écran (`24 x 80`), la fenêtre a la dimension de l'écran.

Exemple:

```
 10 REM TEXTPORT.GSB
 20 HOME:INVERSE:PRINT DATE$;:HPOS=70:PRINT TIME$;:NORMAL:PRINT
 30 HPOS=31:PRINT "Utilisation de TEXTPORT"
 40 TEXTPORT 25,5 TO 60,15:INVERSE:HOME
 50 PRINT "      Utilisation de TEXTPORT      "
 60 PRINT "       TEXTPORT 25,5 TO 60,15      ":PRINT
 70 PRINT "Donne une fenètre de 35 colonnes de"
 80 PRINT "    large sur 11 lignes de haut    ":PRINT
 90 PRINT "  Signifie une fenètre qui débute  ":PRINT
100 PRINT "    Colonne 25, ligne 5 et qui     ":PRINT
110 PRINT "    finit Colonne 60, ligne 15     ";:NORMAL
120 TEXT:END
```

Token: 8C

L'instruction IF détermine si la condition est VRAIE ou FAUSSE. Quand cette condition est vraie (différente de 0), l'instruction ou le renvoi à l'instruction qui suit le THEN est exécuté. Sinon, la ligne suivante est analysée même si elle comporte d'autres instructions.

On peut utiliser une chaîne de caractère comme condition. Le test se fait sur la longueur de la chaîne. Si la chaine est nulle, elle est considérée comme fausse, mais vraie si elle contient un caractère.

On peut avoir un numéro de ligne, un label ou une instruction à la suite du THEN. On peut remplacer THEN par GOTO mais pas par GOSUB.

La suite IF ... THEN ... ELSE peut être sur une, deux ou trois lignes différentes.

Token: DF E2 Variable réservée

Syntaxe:

**1 -** `PRINT TIME$`
**2 -** `TIME$ h,m,s`

`TIME$` est à la fois une variable réservée (syntaxe en 1) et une instruction qui permet de modifier l'heure (syntaxe en 2).

Pour modifier l'heure utiliser trois expressions mathématiques qui ont une valeur comprise entre:

`h` `0` et `23`, pour les heures,
`m` `0` et `59`, pour les minutes,
`s` `0` et `59`, pour les secondes.

Token: DF B9 Fonction

Syntaxe: `PRINT TIME(n)`

`TIME()` permet de lire chaque composante de l'heure au lieu de la lire entièrement comme fait `TIME$`. L'argument `n` doit avoir une valeur comprise entre `0` et `3`. `TIME(0)` initialise la fonction et stocke les valeurs heures, minutes et secondes dans `TIME()` `1` à `3`.

`TIME(0)` initialise la fonction et lit l'heure.
`TIME(1)` lit l'heure,
`TIME(2)` lit les minutes,
`TIME(3)` lit les secondes.

Token: 94

Syntaxe:

```
TIMER ON
TIMER OFF
```

TIMER ON lit l'heure de l'horloge de l'**Apple IIGS**, calcule les secondes écoulées depuis minuit et stocke ce nombre dans le compteur SECONDSà qui est incrémenté à chaque nouvelle seconde. Quand TIMER ON a été exécuté, un programme peut initialiser l'instruction ON TIMER et faire un calcul de durée en interrogeant le contenu de SECONDSà.

Voir la variable réservée SECONDSà.

Exemple:

```
10          REM TIMER.GSB
20          TIMER ON:REM Initialisation du timer
30          B=SECONDSà:REM Sauve la valeur de départ de SECONDSà
35 BOUCLE:  BREAK ON
40          FOR A%=0 TO 10000
50          NEXT
60          C=SECONDSà:REM Récupère la valeur de fin de SECONDSà
70          PRINT "La boucle a durée: "C-B;
80          IF C-B>1 THEN PRINT " secondes":ELSE PRINT " seconde"
90          TIMER OFF
100         PRINT CHR$(7) : REM Signale la fin du programme!
110         BREAK OFF
120 FIN:    END

RUN

La boucle a durée: 7 secondes
```

Token: DF 81

`TO` est utilisé dans la boucle `FOR ... NEXT` ainsi que dans les instructions suivantes: `TEXTPORT, TRACE, TYPE, EDIT.`

Voir ces instructions.

Token: BE

Syntaxe:

```
TRACE
TRACE TO £numfichier
```

TRACE imprime un £ suivi du numéro de chaque ligne pour chaque instruction exécutée.

Quand on utilise l'option TO £numfichier, on adresse les informations TRACE vers le dit numfichier. Il faut bien évidemment avoir préalablement ouvert ce fichier.

Si on redirectionne TRACE, il est préférable de le faire vers l'imprimante ou un *RAM* disque car l'exécution du programme est ralentie lors des accès disque et cela peut provoquer une erreur.

TRACE est désactivé par NOTRACE, LOAD et RUN nomfichier. CHAIN ou RUN ne désactivent pas TRACE.

Message d'erreur possible:

```
?DISK FULL ERROR
```

Token: DF 90

Type de fichier contenant du texte, utilisé entre autres par *APW*, *APPLE WRITER* , *APPLEWORKS* etc...

Utiliser `TXT` lors de la création ou de l'ouverture d'un fichier.

Exemple:

```
OPEN monfichier, FILTYP= TXT AS £1
```

ou bien:

```
CREATE monfichier, FILTYP= TXT
```

Token: DF 99 Fonction

Syntaxe: `A = TYP(numfichier)`

`TYP()` est utilisé pour déterminer le type de la prochaine donnée lue par le **IIGS Basic** dans un fichier de données basic (`BDF`). L'argument `numfichier` de la fonction `TYP` peut être un expression arithmétique mais doit faire référence au fichier ouvert référencé par `£n`; voir `OPEN`.

Le nombre retourné par `TYP()` correspond au type de la donnée qui est lue dans le fichier référencé `numfichier`. Le tableau ci-dessous indique la liste des valeurs possibles et leur signification.

| Valeur | Le prochain caractère à lire sera | Terminateur de variable |
|---|---|---|
| 0 | la fin du fichier | |
| 1 | inutilisé | |
| 2 | un entier simple | % |
| 3 | un entier double | à |
| 4 | un entier long | & |
| 5 | une variable réelle simple | sans terminateur |
| 6 | une variable réelle double | £ |
| 7 | une chaîne de caractères | $ |

Message d'erreur possible:

```
?FILE TYPE ERROR
```

Exemple:

```
 10       REM TYP.GSB (lit le contenu d'un fichier de type BDF)
 20 DEBUT:OPEN "DATA.GSB", FOR INPUT AS £1
 30       FOR I% = 0 TO EOFMARK(1)
 40           B = TYP(1): REM B = Type de la prochaine donnée à lire
 50           ON B GOSUB 90,100,110,120,130,140,150
 60           IF NOT B THEN I%=EOFMARK(1): REM Si B = 0 on a fini
 70       NEXT
 80 FIN:  END
 90       RETURN:REM                          Inutilisé
100       READ£1;A%:PRINTA%:RETURN:REM Entier simple
110       READ£1;Aà:PRINTAà:RETURN:REM Entier double
120       READ£1;A&:PRINTA&:RETURN:REM Entier long
130       READ£1;A:PRINTA:RETURN:REM Variable réelle simple
140       READ£1;A£:PRINTA£:RETURN:REM Variable réelle double
150       READ£1;A$:PRINTA$:RETURN:REM Variable de chaine
```

Token: 9C Commande

Syntaxe: `TYPE nomfichier [TO £n] [,ncar]`

`TYPE` ouvre le fichier `nomfichier` de type `SRC` ou `TXT`, lit les lignes de texte terminées par un retour chariot et affiche les lignes à l'écran.

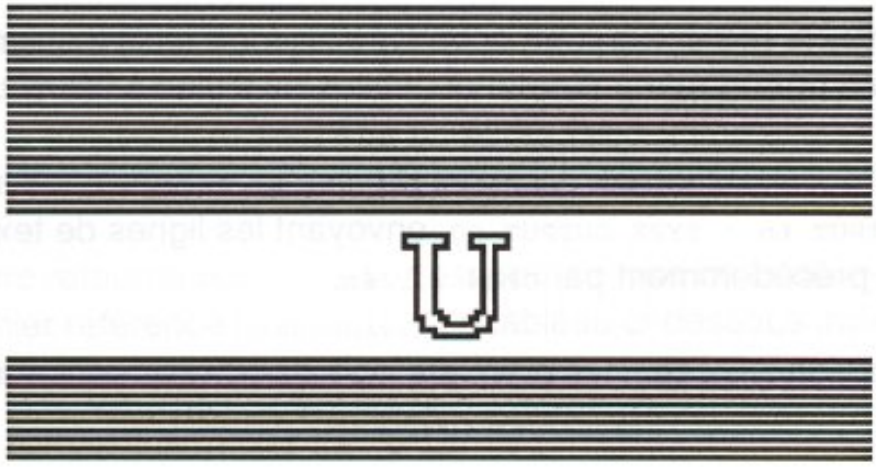
Quand l'option `TO £numfichier` est utilisée, la séquence suivante est exécutée: `OUTPUT £n : TYPE OUTPUT £0`, envoyant les lignes de texte dans le fichier ouvert précédemment par `OPEN ... £n`.

Avec l'option `ncar` seul les premiers `ncar` caractères sont affichés.

Exemple:

```
OPEN .PRINTER, AS £1 : TYPE PICS.TXT, TO £1
```

Dans un premier temps il faut avoir un fichier `PICS.TXT` sur la disquette. Ensuite, ouvrir le périphérique `.PRINTER` et lui attribuer le numéro `£1`. Pour terminer, envoyer sur l'imprimante le contenu du fichier `PICS.TXT`.



UBOUND() ..... 221
UCASE$() ..... 222
UIR() ..... 223
UNLOCK ..... 228
UNTIL ..... 229
UPDATE ..... 230
USING ..... 231

Token: DF DA

Syntaxe: `UBOUND(nompointeur [()] [,numdim])`

`UBOUND()` retourne l'indice supérieur d'un tableau de pointeurs. le paramètre `numdim` est un nombre optionnel qui est utilisé dans les tableaux multidimensionnels. Il spécifie quelle est la dimension du tableau tester.

L'indice supérieur d'un tableau de pointeurs est la plus grande case possible du tableau; l'indice inférieur est toujours 0.

Token: DF C9 Fonction

Syntaxe: `A$ = UCASE$("MinusculEs")`

La fonction `UCASE$()` retourne la chaîne argument après avoir changé toutes les minuscules en majuscules; de `a` à `z` en `A` à `Z`.

Exemple:

```
PRINT UCASE$("MAJusculEs")
MAJUSCULES
```

Token: DF BA

Fonction

Syntaxe:

```
PRINT UIR(n)
PRINT UIR£(n)
```

La fonction `UIR£` retourne le statut de l'information de l'`UIR` après avoir exécuté un `INPUT USING`.

La fonction `UIR()` retourne le statut suivant:

| N° fonction | Retourne l'état de: |
|---|---|
| UIR(0) | Exit. |
| UIR(1) | Valeur ASCII de la dernière touche pressée. |
| UIR(2) | Valeur ASCII masquée de la dernière touche pressée. |
| UIR(3) | Réentrée. |
| UIR(4) | Dernière position x du curseur. |
| UIR(5) | Dernière position y du curseur. |
| UIR(6) | Dernière position relative du curseur. |

UIR: User Input Routine

Syntaxe:

```
10 IMAGE MaxLen,CursorX,CursorY,Scrnwidth,FillChar,CursorMode,Short,Long,
   ModMask,Control,Immediate,Beep,BordChar,Spare,nChars,TChar1,TModfr1,TMode1
  [,TCharn,TModfrn,TModen]
20 PRINT "Question: ":INPUT USING nl;"Réponse"
```

Pour utiliser l'**UIR** (User Input Routine, en français: routine d'entrée définie par l'utilisateur), on doit attribuer des valeurs aux différents paramètres de IMAGE.

Signification des paramètres de l'instruction IMAGE:

- MaxLen — Nombre maximum de caractères de la chaine résultante,
- CursorX — Position horizontale du curseur (entre 1 et 80),
- CursorY — Position verticale du curseur (entre 1 et 24),
- Scrnwidth — Nombre maximum de caractères à afficher à l'écran,
- Fillchar * — Caractère de remplissage,
- CursorMode — Curseur d'insertion (0) ou de remplacement (1),
- Short — Temporisation d'affichage du curseur sur le caractère,
- Long — Temporisation d'affichage du caractère sous le curseur,
- ModMask — Masque de modification (voir table 1),
- Control — Autorise les caractères de controle (1), l'interdit (0)
- Immediate — Normalement à 0, le mettre à 1 pour utilisation avec module externe,
- Beep — 1 = bip si caractère illégal, 0 = pas de beep,
- BordChar * — Affiche le caractère BordChar si bord droit atteint. Si ce caractère est un espace (valeur 32) il est possible d'entrer un nombre de caractères supérieur à la valeur ScrnWidth mais ils disparaitront de l'écran, pour tout autre caractère, ce caractère s'affiche si le bord est atteind et interdit d'entrer d'autres caractères,
- Spare — Doit être à 0, réservé pour une utilisation future,
- NChars — Nombre de caractères spéciaux, 1 par défaut. Un caractère spécial est défini par les trois paramètres suivants:

- TChar1 * — Caractère de fin d'entrée,
- TModfr1 — Quel caractère a été pressé? (voir Table 2),
- TMode1 — 0 = le caractère de fin est un caractère de fin,
  1 = le caractère de fin est un caractère d'interruption,

Il faut définir autant de caractères spéciaux que la valeur de `NChars`; c'est-à-dire `NChars`-1 fois les trois arguments optionnels suivants:

- `TChar n` * ajoute un caractère; identique à `Tchar1`,
- `TModfr n` ajoute un caractère; identique à `TModfr1`,
- `TMode n` ajoute un caractère; identique à `Tmode1`.

Remarques:

- `n` a une valeur comprise entre 2 et 10 dans les trois derniers arguments optionnels,
- tous les arguments ci-dessus peuvent-être remplacés par une variable numérique sauf ceux suivis d'un astérisque,
- il est possible de prévoir une réponse par défaut (voir exemple1),
- il peut y avoir une différence entre `MaxLen` et `ScrnWidth`. Cette différence est fonction de la valeur de `BordChar`. Voir `BordChar`,
- tout caractère défini en `BordChar` reste affiché,
- `FillChar`, `BordChar` et `TChar` doivent avoir une valeur inférieure à 128.

Exemple 1:

```
  5 REM UIR1.GSB
 10 TEXT:HOME:Question$="Question":Reponse$="Réponse"
 20 IMAGE MaxLen,CursorX,CursorY,ScrnWidth,30,CursorMode,Short,Long,ModMask,
    Control,Immediate,Beep,33,Spare,NChars,13,Tmodfr1,Tmode1,27,Tmodfr2, Tmode2
 30 Maxlen=20:CursorX=21:CursorY=1:ScrnWidth=10:CursorMode=0
 40 Short=150:Long=300:ModMask=36:Control=0:Immediate=0:Beep=1:B ordChar=32
 50 Spare=0:NChars=2:Tmodfr1=0:Tmode1=0:Tmodfr2=0:Tmode2=0
 60 LOCATE CursorY,10
 70 PRINT Question$
 80 INPUT USING 20;Reponse$
 90 CursorY=CursorY+2:LOCATE CursorY,10
100 PRINT Question$;"1"
110 INPUT USING 20;Reponse1$
120 LOCATE 5,10:PRINT Reponse$;"    ";Reponse1$
RUN
         Question   Réponse...
         Question1  Réponse1..


         Réponse    Réponse1
```

Lors de l'input, il est possible d'utiliser les options d'édition suivantes:

- <Delete> Efface le caractère à gauche du curseur,
- <Control-E> Passe du curseur de remplacement au curseur d'insertion, et vice-versa,
- <Control-X> Efface le contenu de la ligne actuelle,
- <Control-Y> Efface les caractères depuis le curseur jusqu'à la fin de la ligne,
- <Flèche Gauche> Déplace le curseur d'une colonne vers la gauche,
- <Pomme-Flèche Gauche> Déplace le curseur vers le premier caractère du mot précédent,
- <Flèche droite> Déplace le curseur d'une colonne vers la droite,
- <Pomme-Flèche droite> Déplace le curseur vers le dernier caractère du mot suivant.

Messages d'erreur possibles:

```
?SYNTAX ERROR
?INPUT USING PARM ERROR
```

**TABLE 1:** ModMask

| bit UIR | Valeur | Touche correspondante | Utilisation générale |
|---|---|---|---|
| 7 | 128 | Keypad | 0 (pas reconnu) |
| 6 | 64 | Control | 0 (pas reconnu) |
| 5 | 32 | Option | 1 (reconnu) |
| 4 | 16 | Touche de verrouillage | 1 (reconnu) |
| 3 | 8 | Shift | 1 (reconnu) |
| 2 | 4 | Pomme | 1 (reconnu) |
| 1 | 2 | Etat de Btn0 | 0 (pas reconnu) |
| 0 | 1 | Etat de Btn1 | 0 (pas reconnu) |

**TABLE 2:** TModfr

| bit UIR | Valeur | Touche correspondante | Position demandée |
|---|---|---|---|
| 7 | 128 | Keypad | Touche pressée |
| 6 | 64 | Control | Touche pressée |
| 5 | 32 | Option | Touche pressée |
| 4 | 16 | Touche de verrouillage | Touche pressée |
| 3 | 8 | Shift | Touche pressée |
| 2 | 4 | Pomme | Touche pressée |
| 1 | 2 | Etat de Btn0 | Touche pressée |
| 0 | 1 | Etat de Btn1 | Touche pressée |

## Exemple2:

```
 5       REM UIR2.GSB
10 DEBUT:TEXT:HOME:Question$="Question:":DIM Reponse$(20)
20       Reponse$(0)="Réponse par défaut"
30       IMAGE ML,CX,CY,ML1,30,0,150,200,36,0,0,1,33,0,2,13,0,0,27,0,0
40       ML=20:ML1=20:CX=12:CY=1
50 BCL1: FOR I%=0 TO 20
60           LOCATE CY,1:PRINT Question$
70           INPUT USING 30;Reponse$(I%)
80           CY=CY+1
90       NEXT
100      IF UIR(1)=27 THEN 40
110 BCL2: FOR I%=0 TO 20
120          PRINT Reponse$(I%)
130      NEXT
140 FIN:  END
```

Token: A4 Commande

Syntaxe: `UNLOCK nomfichier`

`UNLOCK` déverrouille la protection à l'écriture/effacement opérée précédemment par `LOCK` et enlève l'astérisque lors de l'affichage du catalogue.

La commande `UNLOCK` rétablit l'affichage des privilèges `DN. W.` lors de l'exécution de la commande `DIR`.

Token: DD Verbe réservé

Syntaxe: `UNTIL [condition]`

`UNTIL` est utilisé conjointement à `DO` et/ou `WHILE` pour créer différents types de boucles conditionnelles.

Le verbe `UNTIL` marque la fin de la boucle. Il est utilisé avec ou sans expression conditionnelle.

Voir les instructions `DO` et `WHILE`.

```
-------------------------------------------------------------------------------
  Combinaison                                                  ! Appellation commune
-------------------------------------------------------------------------------
DO: Instructions :UNTIL Condition                              ! DO . UNTIL
DO: Instructions :WHILE Condition : Instructions :UNTIL        ! DO . WHILE . UNTIL
DO: Instruc :WHILE Condition : Instruc :UNTIL Condition        ! DO . WHILE . UNTIL
WHILE Condition : Instructions :UNTIL                          ! WHILE . WEND
WHILE Condition : Instructions :UNTIL Condition                ! WHILE . UNTIL
WHILE : Instructions :UNTIL condition                          ! REPEAT . UNTIL
WHILE : Instructions :UNTIL                                    ! Boucle infinie
-------------------------------------------------------------------------------
```

Token: DF 8F

Syntaxe: `OPEN fichier, FOR UPDATE AS £n`

`UPDATE` s'utilise conjointement à la commande `OPEN`. Lorsqu'on ouvre un fichier, l'option par défaut est: `OPEN monfichier, FOR UPDATE AS £1. FOR UPDATE` écrit à partir du premier octet du fichier jusqu'à `CLOSE £1. n` a une valeur comprise entre 1 et 29.

Voir l'instruction `OPEN`.

Exemple:

```
 10          REM           FOR.UPDATE.GSB
 20 DEBUT:   File$="/II.GSB/U/FOR.UPDATE.TXT"
 40          OPEN FILE$, FILTYP= TXT  FOR UPDATE  AS#10
 50          FOR A%=0 TO 9
 60              PRINT#10"Ligne: ",A%
 70          NEXT
 80          FOR A%=10 TO 19
 90              PRINT#10"Ligne: ",A%
100          NEXT
110          CLOSE#10
120          TYPE File$
130 FIN:     END
```

Token: DF 83

Syntaxe: voir INPUT, PRINT et UIR.

## 1 - INPUT USING nl

INPUT USING nl exécute la *User Input Routine* en utilisant les paramètres de l'instruction IMAGE. La *User Input Routine* est la même routine qui est utilisée par la commande EDIT.

INPUT USING 0;var$ exécute un INPUT en utilisant les paramètres de la dernière instruction IMAGE utilisée.

## 2 - PRINT USING nl

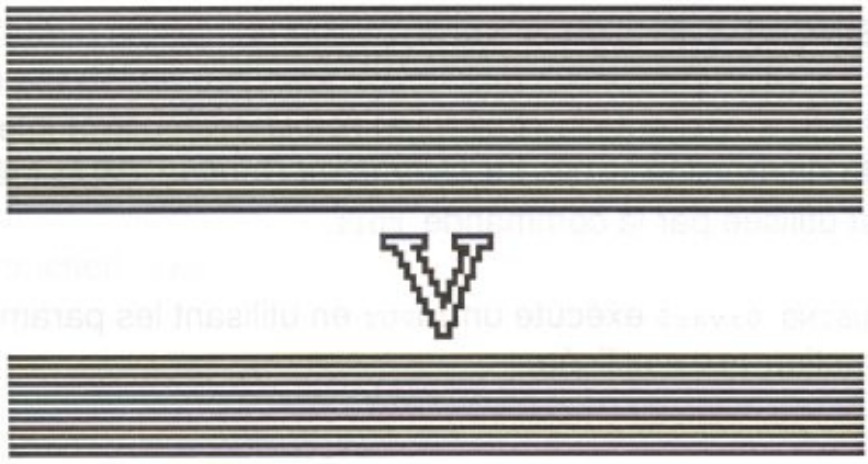
Affiche du texte en utilisant le formatage précisé par l'instruction IMAGE de la ligne nl.

## 3 - PRINT £n USING nl

Affiche du texte sur le périphérique £n en utilisant le formatage précisé par l'instruction IMAGE de la ligne nl.

*User Input Routine* : Routine d'input contrôlée par l'utilisateur.

Voir les instructions IMAGE, PRINT et UIR.



VAL() ..... 233
VAR$() ..... 234
VAR() ..... 235
VARPTR(), VARPTR$() ..... 236
VOLUMES ..... 237
VPOS ..... 238

Token: DF C7 Fonction

Syntaxe: `x = VAL(a$)`

La fonction `VAL()` évalue et retourne la valeur de la chaîne `a$`. La valeur retournée un nombre entier. Si le contenu de la chaîne `a$` n'est pas un nombre, la valeur retournée est nulle (soit: `0`). Une chaîne de plus de `255` caractères occasionne une erreur.

Si la valeur absolue du nombre représentée par la chaîne est supérieure à `1.7E+308`, une erreur se produit.

On ne doit pas utiliser `IF VAL(a$) ... THEN`, `ON VAL(a$) GOTO ...` ou bien `ON VAL(a$) GOSUB n,n,n,n`. Il faut stocker dans une variable `c = VAL(a$)` la valeur de `a$`, puis utiliser `IF c ... THEN...` ou `ON c GOTO ...`

Messages d'erreur possibles:

```
?STRING TOO LONG ERROR
?OVERFLOW ERROR
```

Token: DF D8 Fonction

Syntaxe: `a$ = VAR$(adresse [,ncar])`

La fonction `VAR$()` crée une chaîne de caractères de type *Pascal* à partir du contenu de l'adresse `adresse`. Le contenu du premier octet de `adresse` contient le nombre de caractères à extraire.

Si le paramètre optionnel `ncar` est utilisé, cette chaîne aura `ncar` de long.

La longueur maximale de la chaîne sera donc fonction de la valeur de `ncar` soit entre `0` et `255` caractères.

Le programme `GSB.HELLO` montre une façon d'utiliser la fonction `VAR$()`.

Une chaîne de type *Pascal* est constituée d'un octet qui a pour valeur le nombre de caractères de la chaîne suivit de la chaîne de caractères.

Token: DF D9 Fonction

Syntaxe:

```
VAR(struct,vtype [,longueur])
VAR(adrexpr,type [,longueur])
```

La fonction VAR() extrait une variable d'une structure ou de la mémoire. VAR() peut être utilisée partout où une expression peut l'être à la droite de SET ou de LET.

vtype sera spécifiée en fonction du tableau suivant:

| vtype | La variable sera: | Longueur en nombre d'octets: |
|---|---|---|
| 1 | un réel étendu | |
| 2 | un entier | 1 ou 2 |
| 3 | un double entier | entre 1 et 4 |
| 4 | un entier long | entre 1 et 8 |
| 5 | un simple réel | |
| 6 | un double réel | |
| 7 | une chaîne de caractères | entre 1 et 255 |

La deuxième forme de la fonction VAR() permet l'équivalent d'un PEEK sur plusieurs octets. Le paramètre type est codé de la même façon que vtype dans la première forme, mais correspond en plus au nombre d'octets qui seront lus. Voir tableau ci-dessus.

Token: Fonction

**1 -** `VARPTR(` `DF D6`
**2 -** `VARPTR$(` `DF D7`

Syntaxe:

**1 -** `VARPTR(nomvar)`
**2 -** `VARPTR$(varchaine)`

La fonction `VARPTR()` renvoie l'adresse de la variable argument. Pour les variables de chaîne, `VARPTR()` renvoie l'adresse du descripteur de chaîne. `VARPTR$(varchaine)` retourne l'adresse des données de la variable de chaîne.

`VARPTR$()` retourne une erreur si on lui donne une variable numérique comme argument.

`VARPTR()` retourne une erreur si on lui donne une variable numérique ou un tableau non défini comme argument.

Messages d'erreur possibles:

```
?TYPE MISMATCH ERROR
?VARIABLE ERROR
```

Exemple:

```
 5 REM VARPTR.GSB
10 A$="12":A=21:Aà=210:A&=2100:A£=21000:DIM A!(10)
18 PRINT "-----------------------------------"
20 PRINT "La variable: est stockée à l'adresse:"
22 PRINT "-----------------------------------"
25 PRINT "     A$" TAB(30)"$"+RIGHT$(HEX$(VARPTR(A$)),6)
30 PRINT "     C$" TAB(30)"$"+RIGHT$(HEX$(VARPTR(A)),6)
40 PRINT "     D$" TAB(30)"$"+RIGHT$(HEX$(VARPTR(Aà)),6)
50 PRINT "     E$" TAB(30)"$"+RIGHT$(HEX$(VARPTR(A&)),6)
60 PRINT "     F$" TAB(30)"$"+RIGHT$(HEX$(VARPTR(A£)),6)
70 PRINT "     G$" TAB(30)"$"+RIGHT$(HEX$(VARPTR(A!(0))),6)
80 PRINT "-----------------------------------"
90 END
```

Token: B3 Commande

Syntaxe: VOLUMES

La commande VOLUMES tente de lire le nom de volume des lecteurs de disquettes de .D1 à .D9. Une ligne par périphérique est affichée et contient:

```
.Dn      /nom de volume      nombre de blocks libres
```

de .D1 à .D9. Si le périphérique est vide, le message: DRIVE EMPTY ou DEVICE NOT CONNECTED est affiché à la place du nom de volume et du nombre de blocks libres.

Exemple:

```
VOLUMES
.D1   /HARD.DISK          16812
.D2   /GSB                32
.D3   /UTILITIES          351
.D4   /APW1.0             44
.D5   DRIVE EMPTY
.D6   DRIVE EMPTY
```

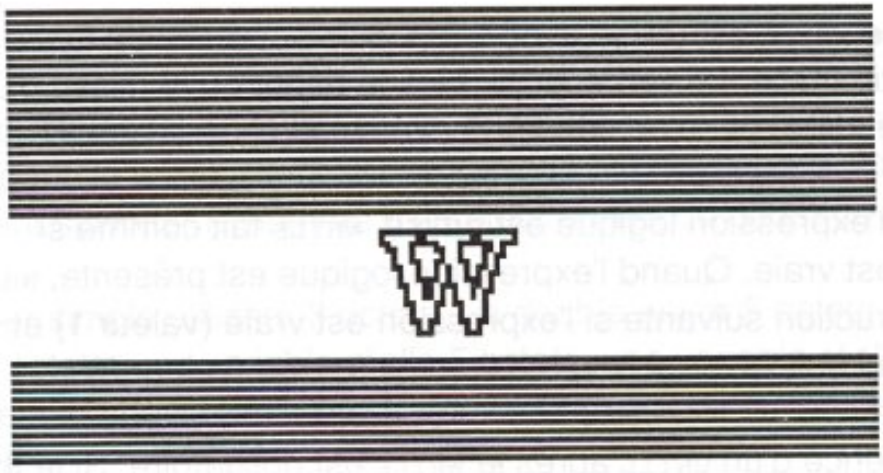
Token: DF E1 Variable

Syntaxe: `VPOS = n`

La variable réservée modifiable `VPOS` contient la position verticale actuelle de l'écriture. On peut modifier le contenu de `VPOS` de façon à écrire le message désiré à la ligne voulue. Donner une valeur supérieure à `255` à `VPOS` provoque une erreur: `?ILLEGAL QUANTITY ERROR`. Assigner à `VPOS` une valeur plus grande que `24` provoque l'inscription du message sur la ligne `24`. La valeur zéro (0) est convertie en un (1).

Exemple:

```
 5 REM VPOS.GSB
10 HOME
15 PRINT "Pomme   VPOS  HPOS"
20 FOR i%=2 TO 18
30   VPOS=i%:HPOS=i%:PRINT "Apple    "VPOS "    "HPOS
40 NEXT
45 VPOS=i%:HPOS=i%:PRINT "Pomme   VPOS  HPOS"
50 END
```



WHILE..... 240
WRITE..... 241

Token: DA — Verbe réservé

Syntaxe: `WHILE [expressionlogique]`

Le verbe WHILE est utilisé avec UNTIL ou DO ... UNTIL de façon à créer une boucle conditionnelle. Le verbe WHILE sera le début ou le milieu d'une boucle et pourra être utilisé avec ou sans une expression conditionnelle.

Quand l'expression logique est omise, WHILE fait comme si l'expression est vraie. Quand l'expression logique est présente, WHILE exécute l'instruction suivante si l'expression est vraie (valeur 1) et saute l'instruction qui suit UNTIL si l'expression est fausse (valeur 0).

La présence d'un UNTIL après le WHILE est obligatoire. Si le **IIGS Basic** ne le trouve pas, il génère une erreur.

| Combinaison | Appellation commune |
|---|---|
| DO: Instructions :UNTIL Condition | DO ... UNTIL |
| DO: Instructions :WHILE Condition : Instructions :UNTIL | DO...WHILE... UNTIL |
| DO: Instruc :WHILE Condition : Instruc :UNTIL Condition | DO . WHILE . UNTIL |
| WHILE Condition : Instructions :UNTIL | WHILE ... WEND |
| WHILE Condition : Instructions :UNTIL Condition | WHILE ... UNTIL |
| WHILE : Instructions :UNTIL condition | REPEAT ... UNTIL |
| WHILE : Instructions :UNTIL | Boucle infinie |

Voir également DO et UNTIL.

Message d'erreur possible:

```
?WHILE w/o UNTIL
```

Token: AD Commande

Syntaxe:

```
WRITE £numfichier [,numrec] [;var [,var$]]
```

`WRITE £n` écrit séquentiellement la valeur de chacune de ses expressions dans le fichier de données spécifié par `numfichier`. On peut éventuellement spécifier le numéro d'enregistrement du début de stockage.

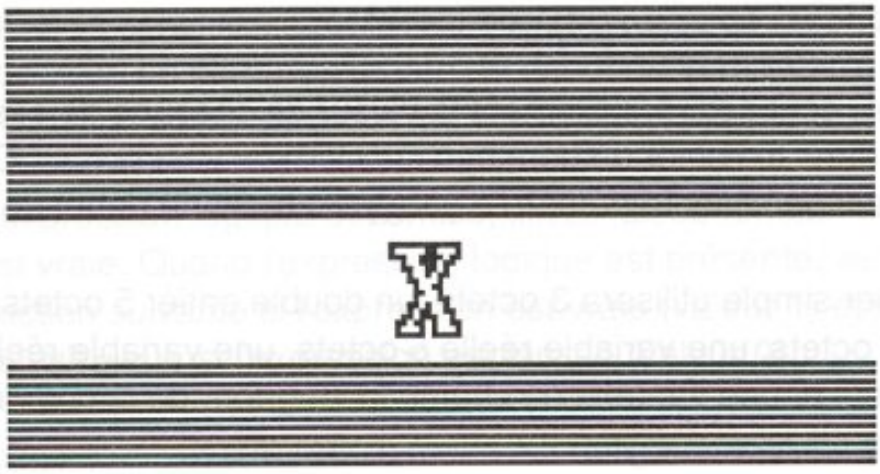
Un entier simple utilisera 3 octets, un double entier 5 octets, un entier long 9 octets, une variable réelle 5 octets, une variable réelle double 9 octets et une variable de chaîne la longueur de la chaîne plus 2 octets.

| Type de variable | Terminateur | Exemple |
|---|---|---|
| Réelle simple | Sans | Nombre |
| Réelle double | £ | Nombre£ |
| Structure | !(n) | Nombre!(0) |
| Entier simple | % | Nombre% |
| Entier double | à | Nombreà |
| Entier long | & | Nombre& |
| Chaîne de caractère | $ | Nombre$ |

Exemples:

```
 10         REM   WRITE.GSB
 20 DEBUT:  Fichier$="READ.BDF"
 30         Ligne$="Ligne numéro "
 40         OPEN Fichier$, FILTYP= BDF  AS£1
 50 BOUCLE:BREAK ON
 60         FOR X%=1 TO 10
 70             WRITE£1;Ligne$,X%
 80             PRINT Ligne$;X%
 90         NEXT
100         CLOSE£1
110 FIN:    END
```

```
10 REM    WRITE1.GSB
20 FILE$="READ1.BDF"
40 OPEN FILE$, FILTYP= BDF  FOR OUTPUT AS£12
50 A%=12:Aà=24:A&=48:A=96:A£=192:A$="384"
60 WRITE£12;"Test d'écriture des formats de variables: A%,Aà,A&,A,A£,A$"
,A%,Aà ,A&,A,A£,A$
80 CLOSE£12
90 END
```



XOR..... 243

Token: DF 8A

Opérateur logique

Syntaxe: `x = m XOR n`

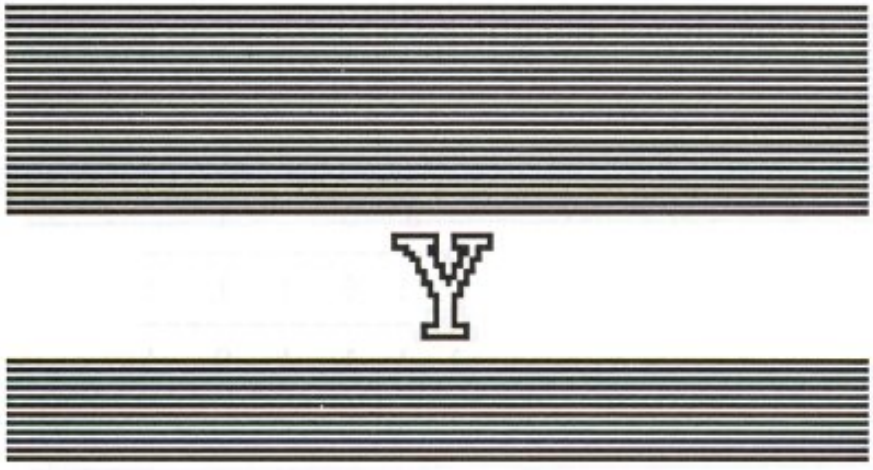
L'opérateur logique XOR retourne le OU EXCLUSIF de ses deux arguments.

Table de vérité

| b \ a | 0 | 1 |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 1 | 0 |

Exemple:

```
PRINT 3 XOR 0
1
```



# Annexes

Erreurs GSBasic..... 247
Erreurs ProDOS..... 249
Table ASCII de $00 à $1F..... 250
Table ASCII de $20 à $3F..... 251
Table ASCII de $40 à $5F..... 252
Table ASCII de $60 à $7F..... 253
Table ASCII de $80 à $9F..... 254
Table ASCII de $A0 à $BF..... 255
Table ASCII de $C0 à $DF..... 256
Table ASCII de $E0 à $FF..... 257
Anomalies du GSBasic..... 258

| N° | Message d'erreur | Texte français |
|---|---|---|
| | | |
| 0 | | Pas d'erreur |
| 1 | NEXT w/o FOR | NEXT sans FOR |
| 2 | SYNTAX | Erreur de syntaxe |
| 3 | RETURN w/o GOSUB | RETURN sans GOSUB |
| 4 | OUT of DATA | Manque des DATA |
| 5 | ILLEGAL QUANTITY | Quantité illégale |
| 6 | INVALID DATA | DATA invalide |
| 7 | ILLEGAL LINE NUMBER/LABEL | Numéro de ligne ou label illégal |
| 8 | DUPLICATE LABEL | Label en double |
| 9 | OVERFLOW | Résultat trop grand |
| 10 | OUT of MEMORY | Erreur afférente à la mémoire |
| 11 | UNDEF'D STATEMENT | Numéro de ligne indéterminé |
| 12 | BAD SUBSCRIPT | Numéro de dimension hors tableau |
| 13 | RANGE | Nombre trop grand |
| 14 | STACK OVERFLOW | Débordement de pile |
| 15 | DUPLICATE DEFINITION | Définition en double |
| 16 | DIVISION by ZERO | Division par zéro |
| 17 | ILLEGAL DIRECT | Commande à utiliser dans un programme |
| 18 | TYPE MISMATCH | Type de variables incompatible |
| 19 | STRING TOO LONG | Chaîne trop longue |
| 20 | FORMULA TOO COMPLEX | Formule trop complexe |
| 21 | CAN'T CONTINUE | Je ne peux pas continuer |
| 22 | UNDEF'D PROC/FUNCTION | Procédure ou Fonction inexistante |
| 23 | VARIABLE | Erreur de variable |
| 24 | TOOLSET CALL | Erreur d'appel outil |
| 25 | ProDOS CALL | Erreur d'appel ProDOS |
| 26 | FILE OPEN | Fichier ouvert |
| 27 | VOLUME TYPE | Erreur de type de volume |
| 28 | DRIVE EMPTY | Lecteur de disquettes vide |
| 29 | FILE TYPE | Erreur de type de fichier |
| 30 | I/O | Erreur d'entrée/sortie |
| 31 | FILE TOO LARGE | Fichier trop grand |
| 32 | WRITE PROTECT | Disquette protégé contre l'écriture |
| 33 | VOLUME SWITCHED | La disquette a été changée de lecteur |
| 34 | BAD PATH | Mauvais nom de fichier |
| 35 | FILE NOT FOUND | Fichier pas trouvé |
| 36 | PATH NOT FOUND | Nom pas trouvé |
| 37 | VOLUME NOT FOUND | Volume pas trouvé |
| 38 | DUPLICATE FILE | Nom de fichier déjà existant |
| 39 | DISK FULL | Disquette pleine |
| 40 | FILE LOCKED | Fichier verrouillé |
| 41 | FILE NOT OPEN | Fichier fermé |
| 42 | DEVICE NOT CONNECTED | Périphérique pas connecté |
| 43 | INT/FCB/VCB TBL FULL | Tables pleines |
| 44 | DIRECTORY FULL | Directory plein |
| 45 | DUPLICATE VOLUME | Nom de volume en double |
| 46 | = ADRS: INTERNAL | Erreur interne |
| 47 | FOR w/o NEXT | FOR sans NEXT |
| 48 | POSITION RANGE | Erreur dans la position |
| 49 | FILE CREATE | Erreur à la création du fichier |
| | | |

| N° | Message d'erreur | |
|---|---|---|
| 50 | DIFFERENT VOLUMES | Différents volumes |
| 51 | DAMAGED DIRECTORY | Directory endommagé |
| 52 | LINE TOO LONG | Ligne trop longue |
| 53 | RESERVED WORD | Mot réservé |
| 54 | ARGUMENT COUNT | Erreur dans le nombre d'argument |
| 55 | ARGUMENT TYPE MISMATCH | Type d'argument erronné |
| 56 | UNDEFINED ARRAY | Tableau non défini |
| 57 | WHILE w/o UNTIL | WHILE sans UNTIL |
| 58 | UNTIL w/o WHILE | UNTIL sans WHILE |
| 59 | MULTI STATEMENT | Erreur dans plusieurs instructions |
| 60 | MISSING OPERAND | Opérande oublié |
| 61 | NESTED DEF | Définition sans argument |
| 62 | RECURSION | Erreur de récursion |
| 63 | MISSING END PROC/FN | Manque END PROC/FN |
| 64 | END PROC/FN w/o DEF | END PROC/FN sans définition |
| 65 | MISSING RETURN | Manque un RETURN |
| 66 | DAMAGE REPORT SCOTTY? | Rapport de dommage Mr. Scott! |
| 67 | NOT LOCAL | Pas une variable locale |
| 68 | MULTI-LINE FN REF | Référence à une fontion multiligne |
| 69 | PROC NAME | Erreur du nom de procédure |
| 70 | INPUT USING PARM | Erreur dans un paramètre INPUT USING |
| 71 | UNCLAIMED EVENT | Evènement non réclamé |
| 72 | SANE INVALID | Valeur trop grande pour SANE |
| 73 | INEXACT | Erreur inexacte (voir SANE) |
| 74 | UNDERFLOW | Résultat trop faible |
| 75 | NOT A NUMBER | N'est pas un nombre |
| 76 | NO SEGMENT | Pas de segment |
| 77 | NO LIBRARY/INVOKE | Pas de LIBRARY/INVOKE |
| 78 | STRING SPACE | Erreur d'espace pour une chaîne |
| 79 | MENU or EVENT | Erreur Menu manager ou Event manager |
| 80 | TASKPOLL INIT & WIND STARTUP | Window manager pas initialisé |
| 81 | DEVICE NOT FOUND | Périphérique pas trouvé |
| 82 | INVALID DEVICE | Périphérique invalide |
| 83 | ProDOS VERSION | Version de ProDOS incompatible |
| 84 | RETURN 0 | Erreur de RETURN 0 |
| 85 | DIR PATTERN | Erreur dans les arguments de DIR |
| 86 | CAN'T RENUMBER | Je ne peux pas renuméroter |
| 87 | STRING TEMP OVERFLOW | Débordement de variables |
| 88 | CLOSED FCB | FCB fermé |
| 89 | NOT IMPLEMENTED | Erreur pas implémentée |
| | PROGRAM ERROR | Erreur du programme |
| | Please Press SPACE BAR | Pressez la barre d'espace |
| | PROGRAM INTERRUPTED | Le programme a été interrompu |
| 253 | EXTRA INPUT IGNORED | Caractères ignorés dans un INPUT |
| 254 | ?REENTER | Ré-entrez l'argument |

| N° $ | Message | Description |
|---|---|---|
| | | |
| 00 | Pas d'erreur | |
| 01 | Invalid call number | Commande inexistante |
| | Unclaimed interrupt | Interruption pas prise en charge |
| 07 | ProDOS is busy | ProDOS est occupé |
| 0A | VCB unusable | La table VCB est endommagée |
| 0B | FCB unusable | La table FCB est endommagée |
| 0C | Block zero allocated illegally | Essai d'écriture sur le block zéro |
| 0D | Interrupt occured while I/O shadowing off | Interruption avec "shadowing off" |
| 10 | Device not found | Périphérique pas trouvé |
| 11 | Invalid device name or number | Nom ou n° de disque pas valable |
| 25 | Interrupt vector table full | Table des interruptions pleine |
| 27 | I/O error | Erreur d'entrée sortie |
| 28 | No device connected | Pas de périphérique branché |
| 2B | Write protected | Disquette protégée en écriture |
| 2E | Disk switched | Le disque à été changé de lecteur |
| 2F | Device not online | Périphérique pas connecté |
| 30 | Device | Erreurs |
| 31 | specific | particulière |
| à | ... | aux périphériques |
| 3F | errors | |
| 40 | Invalid pathname syntax | Erreur de syntaxe du nom du fichier |
| 41 | File type error | Erreur sur le type de fichier |
| 42 | FCB table full | La table FCB est pleine |
| 43 | Invalid file reference number | N° de référence de fichier invalide |
| 44 | Path not found | Nom pas trouvé |
| 45 | Volume not found | Volume pas trouvé |
| 46 | File not found | Fichier pas trouvé |
| 47 | Duplicate pathname | Nom de fichier déjà attribué |
| 48 | Volume full | Disquette pleine |
| 49 | Volume directory full | Table des noms de fichiers pleine |
| 4A | Version error | Version ProDOS incompatible |
| 4B | Unsupported (or incorrect) storage type | Type de stockage de fichier pas reconnu |
| 4C | End Of File encountered | Fin du fichier rencontrée |
| 4D | Position out of range | Eofmark incorrect |
| 4E | Access not allowed | Accès interdit |
| 50 | File is open | Fichier ouvert |
| 51 | Directory structure damaged | Structure du catalogue endommagée |
| 52 | Unsupported volume type | Type de volume pas supporté |
| 53 | Parameter out of range | Valeur d'un paramètre trop grand |
| 54 | Out of memory | Programme ProDOS 8 trop grand |
| 55 | VCB table full | Table VCB pleine |
| 57 | Duplicate volume | Nom de disquette en double |
| 58 | Not a block device | Ce n'est pas un lecteur de disques |
| 59 | Invalid level | La valeur de SET_LEVEL est invalide |
| 5A | Block number out of range | Numéro de block hors capacité disque |
| 5B | Illegal pathname change | Essai de renommer un fichier interdit |
| 5C | Not an executable system file | Fichier spécifié n'est ni $B3, ni $FF |
| 5D | Operating system not available | ProDOS 8 est absent |
| 61 | End of Directory error | Fin de directory |
| | | |

| Hexa | Déci | Binaire | Caractère | Commentaire | |
|---|---|---|---|---|---|
| $00 | 00 | 00000000 | Ctrl-@ | Null | (Ctrl-à) |
| $01 | 01 | 00000001 | Ctrl-A | | |
| $02 | 02 | 00000010 | Ctrl-B | | |
| $03 | 03 | 00000011 | Ctrl-C | Arrête l'exécution du programme | |
| $04 | 04 | 00000100 | Ctrl-D | | |
| $05 | 05 | 00000101 | Ctrl-E | | |
| $06 | 06 | 00000110 | Ctrl-F | | |
| $07 | 07 | 00000111 | Ctrl-G | Bip | (BELL) |
| $08 | 08 | 00001000 | Ctrl-H | <- | (BS) |
| $09 | 09 | 00001001 | Ctrl-I | -> | (HT) |
| $0A | 10 | 00001010 | Ctrl-J | Avance d'une ligne | (LF) |
| $0B | 11 | 00001011 | Ctrl-K | Flèche haut | (VT) |
| $0C | 12 | 00001100 | Ctrl-L | Avance d'une page | (FF) |
| $0D | 13 | 00001101 | Ctrl-M | Touche Return | (CR) |
| $0E | 14 | 00001110 | Ctrl-N | | |
| $0F | 15 | 00001111 | Ctrl-O | | |
| $10 | 16 | 00010000 | Ctrl-P | | |
| $11 | 17 | 00010001 | Ctrl-Q | | |
| $12 | 18 | 00010010 | Ctrl-R | | |
| $13 | 19 | 00010011 | Ctrl-S | | |
| $14 | 20 | 00010100 | Ctrl-T | | |
| $15 | 21 | 00010101 | Ctrl-U | | |
| $16 | 22 | 00010110 | Ctrl-V | | |
| $17 | 23 | 00010111 | Ctrl-W | | |
| $18 | 24 | 00011000 | Ctrl-X | Efface la ligne lors de l'édition | |
| $19 | 25 | 00011001 | Ctrl-Y | | |
| $1A | 26 | 00011010 | Ctrl-Z | | |
| $1B | 27 | 00011011 | Ctrl-[ | Touche Escape (ESC) | (Ctrl-°) |
| $1C | 28 | 00011100 | Ctrl-\ | | (Ctrl-ç) |
| $1D | 29 | 00011101 | Ctrl-] | | (Ctrl-§) |
| $1E | 30 | 00011110 | Ctrl-^ | | |
| $1F | 31 | 00011111 | Ctrl-_ | | |

| Hexa | Déci | Binaire | Caractère | Commentaire |
|---|---|---|---|---|
| | | | | |
| $20 | 32 | 00100000 | Espace | |
| $21 | 33 | 00100001 | ! | |
| $22 | 34 | 00100010 | " | |
| $23 | 35 | 00100011 | # | (£) |
| $24 | 36 | 00100100 | $ | |
| $25 | 37 | 00100101 | % | |
| $26 | 38 | 00100110 | & | |
| $27 | 39 | 00100111 | ' | |
| $28 | 40 | 00101000 | ( | |
| $29 | 41 | 00101001 | ) | |
| $2A | 42 | 00101010 | * | |
| $2B | 43 | 00101011 | + | |
| $2C | 44 | 00101100 | , | |
| $2D | 45 | 00101101 | - | |
| $2E | 46 | 00101110 | . | |
| $2F | 47 | 00101111 | / | |
| | | | | |
| $30 | 48 | 00110000 | 0 | |
| $31 | 49 | 00110001 | 1 | |
| $32 | 50 | 00110010 | 2 | |
| $33 | 51 | 00110011 | 3 | |
| $34 | 52 | 00110100 | 4 | |
| $35 | 53 | 00110101 | 5 | |
| $36 | 54 | 00110110 | 6 | |
| $37 | 55 | 00110111 | 7 | |
| $38 | 56 | 00111000 | 8 | |
| $39 | 57 | 00111001 | 9 | |
| $3A | 58 | 00111010 | : | |
| $3B | 59 | 00111011 | ; | |
| $3C | 60 | 00111100 | < | |
| $3D | 61 | 00111101 | = | |
| $3E | 62 | 00111110 | > | |
| $3F | 63 | 00111111 | ? | |
| | | | | |

| Hexa | Déci | Binaire | Caractère | Commentaire |
|---|---|---|---|---|
| $40 | 64 | 01000000 | @ | (à) |
| $41 | 65 | 01000001 | A | |
| $42 | 66 | 01000010 | B | |
| $43 | 67 | 01000011 | C | |
| $44 | 68 | 01000100 | D | |
| $45 | 69 | 01000101 | E | |
| $46 | 70 | 01000110 | F | |
| $47 | 71 | 01000111 | G | |
| $48 | 72 | 01001000 | H | |
| $49 | 73 | 01001001 | I | |
| $4A | 74 | 01001010 | J | |
| $4B | 75 | 01001011 | K | |
| $4C | 76 | 01001100 | L | |
| $4D | 77 | 01001111 | M | |
| $4E | 78 | 01001110 | N | |
| $4F | 79 | 01001111 | O | |
| $50 | 80 | 01010000 | P | |
| $51 | 81 | 01010001 | Q | |
| $52 | 82 | 01010010 | R | |
| $53 | 83 | 01010011 | S | |
| $54 | 84 | 01010100 | T | |
| $55 | 85 | 01010101 | U | |
| $56 | 86 | 01010110 | V | |
| $57 | 87 | 01010111 | W | |
| $58 | 88 | 01011000 | X | |
| $59 | 89 | 01011001 | Y | |
| $5A | 90 | 01011010 | Z | |
| $5B | 91 | 01011011 | [ | (°) |
| $5C | 92 | 01011100 | \ | (ç) |
| $5D | 93 | 01011101 | ] | (§) |
| $5E | 94 | 01011110 | ^ | |
| $5F | 95 | 01011111 | _ | |

| Hexa | Déci | Binaire | Caractère | Commentaire |
|---|---|---|---|---|
| $60 | 96 | 01100000 | ` | |
| $61 | 97 | 01100001 | a | |
| $62 | 98 | 01100010 | b | |
| $63 | 99 | 01100011 | c | |
| $64 | 100 | 01100100 | d | |
| $65 | 101 | 01100101 | e | |
| $66 | 102 | 01100110 | f | |
| $67 | 103 | 01100111 | g | |
| $68 | 104 | 01101000 | h | |
| $69 | 105 | 01101001 | i | |
| $6A | 106 | 01101010 | j | |
| $6B | 107 | 01101011 | k | |
| $6C | 108 | 01101100 | l | |
| $6D | 109 | 01101101 | m | |
| $6E | 110 | 01101110 | n | |
| $6F | 111 | 01101111 | o | |
| $70 | 112 | 01110000 | p | |
| $71 | 113 | 01110001 | q | |
| $72 | 114 | 01110010 | r | |
| $73 | 115 | 01110011 | s | |
| $74 | 116 | 01110100 | t | |
| $75 | 117 | 01110101 | u | |
| $76 | 118 | 01110110 | v | |
| $77 | 119 | 01110111 | w | |
| $78 | 120 | 01111000 | x | |
| $79 | 121 | 01111001 | y | |
| $7A | 122 | 01111010 | z | |
| $7B | 123 | 01111011 | { | (é) |
| $7C | 124 | 01111100 | \| | (ù) |
| $7D | 125 | 01111101 | } | (è) |
| $7E | 126 | 01111110 | ~ | (¨) |
| $7F | 127 | 01111111 | Delete | |

| Hexa | Déci | Binaire | Caractère | Commentaire |
|---|---|---|---|---|
| $80 | 128 | 10000000 | Ä | |
| $81 | 129 | 10000001 | Å | |
| $82 | 130 | 10000010 | Ç | |
| $83 | 131 | 10000011 | É | |
| $84 | 132 | 10000100 | Ñ | |
| $85 | 133 | 10000101 | Ö | |
| $86 | 134 | 10000110 | Ü | |
| $87 | 135 | 10000111 | á | |
| $88 | 136 | 10001000 | à | |
| $89 | 137 | 10001001 | â | |
| $8A | 138 | 10001010 | ä | |
| $8B | 139 | 10001011 | ã | |
| $8C | 140 | 10001100 | å | |
| $8D | 141 | 10001101 | ç | |
| $8E | 142 | 10001110 | é | |
| $8F | 143 | 10001111 | è | |
| $90 | 144 | 10010000 | ê | |
| $91 | 145 | 10010001 | ë | |
| $92 | 146 | 10010010 | í | |
| $93 | 147 | 10010011 | ì | |
| $94 | 148 | 10010100 | î | |
| $95 | 149 | 10010101 | ï | |
| $96 | 150 | 10010110 | ñ | |
| $97 | 151 | 10010111 | ó | |
| $98 | 152 | 10011000 | ò | |
| $99 | 153 | 10011001 | ô | |
| $9A | 154 | 10011010 | ö | |
| $9B | 155 | 10011011 | õ | |
| $9C | 156 | 10011100 | ú | |
| $9D | 157 | 10011101 | ù | |
| $9E | 158 | 10011110 | û | |
| $9F | 159 | 10011111 | ü | |

| Hexa | Déci | Binaire | Caractère | Commentaire |
|---|---|---|---|---|
| $A0 | 160 | 10100000 | † | |
| $A1 | 161 | 10100001 | ° | |
| $A2 | 162 | 10100010 | ¢ | |
| $A3 | 163 | 10100011 | £ | |
| $A4 | 164 | 10100100 | § | |
| $A5 | 165 | 10100101 | | |
| $A6 | 166 | 10100110 | ¶ | |
| $A7 | 167 | 10100111 | ß | |
| $A8 | 168 | 10101000 | ® | |
| $A9 | 169 | 10101001 | © | |
| $AA | 170 | 10101010 | ™ | |
| $AB | 171 | 10101011 | ´ | |
| $AC | 172 | 10101100 | ¨ | |
| $AD | 173 | 10101101 | ≠ | |
| $AE | 174 | 10101110 | Æ | |
| $AF | 175 | 10101111 | Ø | |
| $B0 | 176 | 10110000 | ∞ | |
| $B1 | 177 | 10110001 | ± | |
| $B2 | 178 | 10110010 | ≤ | |
| $B3 | 179 | 10110011 | ≥ | |
| $B4 | 180 | 10110100 | ¥ | |
| $B5 | 181 | 10110101 | µ | |
| $B6 | 182 | 10110110 | ∂ | |
| $B7 | 183 | 10110111 | Σ | |
| $B8 | 184 | 10111000 | Π | |
| $B9 | 185 | 10111001 | π | |
| $BA | 186 | 10111010 | ∫ | |
| $BB | 187 | 10111011 | ª | |
| $BC | 188 | 10111100 | º | |
| $BD | 189 | 10111101 | Ω | |
| $BE | 190 | 10111110 | æ | |
| $BF | 191 | 10111111 | ø | |

| Hexa | Déci | Binaire | Caractère | Commentaire |
|---|---|---|---|---|
| $C0 | 192 | 11000000 | ¿ | |
| $C1 | 193 | 11000001 | ¡ | |
| $C2 | 194 | 11000010 | ¬ | |
| $C3 | 195 | 11000011 | √ | |
| $C4 | 196 | 11000100 | ƒ | |
| $C5 | 197 | 11000101 | ≈ | |
| $C6 | 198 | 11000110 | Δ | |
| $C7 | 199 | 11000111 | « | |
| $C8 | 200 | 11001000 | » | |
| $C9 | 201 | 11001001 | … | |
| $CA | 202 | 11001010 | | Espace dur |
| $CB | 203 | 11001011 | À | |
| $CC | 204 | 11001100 | Ã | |
| $CD | 205 | 11001111 | Õ | |
| $CE | 206 | 11001110 | Œ | |
| $CF | 207 | 11001111 | œ | |
| $D0 | 208 | 11010000 | – | |
| $D1 | 209 | 11010001 | — | |
| $D2 | 210 | 11010010 | " | |
| $D3 | 211 | 11010011 | " | |
| $D4 | 212 | 11010100 | ' | |
| $D5 | 213 | 11010101 | ' | |
| $D6 | 214 | 11010110 | + | |
| $D7 | 215 | 11010111 | ◊ | |
| $D8 | 216 | 11011000 | Ÿ | |
| $D9 | 217 | 11011001 | | |
| $DA | 218 | 11011010 | | |
| $DB | 219 | 11011011 | | |
| $DC | 220 | 11011100 | | |
| $DD | 221 | 11011101 | | |
| $DE | 222 | 11011110 | | |
| $DF | 223 | 11011111 | | |

| Hexa | Déci | Binaire | Caractère | Commentaire |
|---|---|---|---|---|
| $E0 | 224 | 11100000 | | |
| $E1 | 225 | 11100001 | | |
| $E2 | 226 | 11100010 | | |
| $E3 | 227 | 11100011 | | |
| $E4 | 228 | 11100100 | | |
| $E5 | 229 | 11100101 | | |
| $E6 | 230 | 11100110 | | |
| $E7 | 231 | 11100111 | | |
| $E8 | 232 | 11101000 | | |
| $E9 | 233 | 11101001 | | |
| $EA | 234 | 11101010 | | |
| $EB | 235 | 11101011 | | |
| $EC | 236 | 11101100 | | |
| $ED | 237 | 11101101 | | |
| $EE | 238 | 11101110 | | |
| $EF | 239 | 11101111 | | |
| $F0 | 240 | 11110000 | | |
| $F1 | 241 | 11110001 | | |
| $F2 | 242 | 11110010 | | |
| $F3 | 243 | 11110011 | | |
| $F4 | 244 | 11110100 | | |
| $F5 | 245 | 11110101 | | |
| $F6 | 246 | 11110110 | | |
| $F7 | 247 | 11110111 | | |
| $F8 | 248 | 11111000 | | |
| $F9 | 249 | 11111001 | | |
| $FA | 250 | 11111010 | | |
| $FB | 251 | 11111011 | | |
| $FC | 252 | 11111100 | | |
| $FD | 253 | 11111101 | | |
| $FE | 254 | 11111110 | | |
| $FF | 255 | 11111111 | | |

1 - Problème du RENUM:
Une ligne qui contient l'instruction: INT(.1) provoque le message:

```
0
?SYNTAX ERROR IN 0
```

2 - L'affichage du catalogue avec DIR et CATALOG augmente d'une minute le champ date (17:59 donne à l'écran: 17:60). De plus, une date incorrecte est affichée au lieu de: <No Date>

3 - Envoyer la commande DIR vers l'imprimante provoque le passage en caractères double hauteur sur Image Writer II (non filtrage des icones).
Ligne à éviter:

```
OPEN .PRINTER, AS £1 : OUTPUT £1 : DIR : OUTPUT £0 : CLOSE £1
```

4 - Il se produit une I/O ERROR s'il y a plus de 12 fichiers dans le directory actuel et si on exécute les instructions de la ligne suivante:

```
OPEN FILENAME, FILTYP=TXT AS £1 : OUTPUT £1 : CATALOG : CLOSE £1
```

5 - Problème de l'APPEND. L'écriture des données sur disque avec APPEND ne fonctionne pas comme prévu. En effet, les données sont écrites à partir du 3° octet du dernier block du fichier au lieu de débuter l'écriture à la position EOFMARK+1.

6 - Il y a un problème avec les fichiers à accès direct lié à l'utilisation de

```
CREATE
OPEN MONFICHIER, FILTYP=TXT AS £1,16 : REM Ceci donne un bon résultat.
CREATE MONFICHIER, FILTYP=TXT,16 : REM     Celà donne un mauvais résultat.
```

7 - PRINT USING Il y a un décalage d'un caractère lors de l'affichage et une mauvaise écriture des données sur disquette.

8 - FN l'utilisation des variables de chaîne dans une fonction provoque plus ou moins rapidement une saturation de la mémoire car la mémoire occupée par les variables inutilisées n'est pas libérée; utiliser plutôt une procédure.

9 - Problème lié à la commande INIT: il est impossible en mode immédiat de formater une disquette déjà formatée ou qui utilise un système d'exploitation autre que *ProDOS* .

10 - Problème lié à GET$ x$: la valeur stockée dans x$ sera toujours supérieure à 127.

11 - Problème mathématique:

```
I% = 43 : J% = I% / 2 : PRINT J% : REM Le résultat est: 22    -> FAUX
I% = 43 : J% = I% DIV 2 : PRINT J% : REM Le résultat est: 21 -> JUSTE
```

12 - Il n'y a pas de contrôle sur le numéro de ligne: il est impératif de ne pas entrer une valeur supérieure au maximum autorisé.

Ci-dessous quelques anomalies rencontrées dans le manuel:

- Chapitre 1 (Introduction to Apple IIGS Basic) page 1:
Les paragraphes entre *Syntax checking* et *Starting and stopping programs* exclus sont notés, mais ne figurent pas dans le manuel:

- Chapitre 7 (Advanced topics) page 160, il faut lire:
mettre le paramètre *Beep* à 1 pour qu'**UIR** envoie un beep à chaque caractère illégal.

- Chapitre 8 (Basic notation) page 221 dernière ligne il faut lire:

```
Press Y if you want to destroy /volname
```

- Chapitre 8 (Basic notation) page 244:
L'exemple qui illustre OUTREC contient deux erreurs. En effet:
`OPEN PROGNAM$+".TXT"` envoie le message `?BAD PATH ERROR`
et il manque: `,FILTYP=TXT`.

- Annexe B, page 302/303; le message N°

66 est: `DAMAGE REPORT SCOTTY?`
`CAP'N, THE LITTERAL POOL IS COMPLETELY FRIED!`
87 est: `STRING TEMP OVERFLOW`
88 est: `CLOSED FCB`
89 est: `NOT IMPLEMENTED`

- Il est fait mention de la possibilité de récupérer le catalogue comme en Applesoft (avec INPUT), mais ceci ne fonctionne pas.
- Aucune référence à l'instruction DBUG qui a pour token BA.
- INDENT a quelque problème lorsque FOR ... NEXT est sur une seule ligne,
- Il n'est pas possible d'ouvrir un fichier BDF en utilisant le mot DATA:

```
OPEN FICHIER.BDF, FILTYP=DATA AS £1:REM Cette ligne génère une erreur
OPEN FICHIER.BDF, FILTYP=BDF AS £1:REM Cette ligne est exacte
```

Toutes ces anomalies ont été signalés au **GSBASIC Product Management**.

Achevé d'imprimer
sur les presses de l'imprimerie CITÉ-PRESS
4, rue de la Cour des Noues — 75020 PARIS
Tél. : (16-1) 46.36.85.10
Dépôt légal : Avril 1988

N° d'impression : 261
N° d'édition : 630
N° d'ISBN : 2-901124-27-5