un logiciel

GFA SYSTEMTECHNIK

MICRO APPLICATION

BASIC GFA

UN BASIC SIMPLE, RAPIDE ET PUISSANT POUR ATARI ST[F]

ATARI ST

un logiciel

GFA SYSTEMTECHNIK

MICRO APPLICATION

BASIC GFA

UN BASIC SIMPLE, RAPIDE ET PUISSANT POUR ATARI ST[F]

ATARI ST

Distribué par : MICRO APPLICATION
13, Rue Sainte Cécile
75009 PARIS

ISBN : 2-86899-071-1

Traduction Française assurée par Pierre BRONNENKANT

Collection dirigée par Mr Philippe OLIVIER
Edition réalisée par Frédérique BEAUDONNET

BASIC GfA

- Une très grande vitesse d'exécution.
- Une précision de 11 chiffres significatifs.
- Une programmation structurée.
- Le langage le plus facile pour programmer GEM.
- Un éditeur confortable.

Auteur du livre : Hendrika Hilchner

Auteur du logiciel : Frank Ostrowski

GfA

Systemtechnik

Interpréteur BASIC GfA

Manuel d'utilisation

SOMMAIRE

CHAPITRE I :

Avantages par rapport aux interpréteurs BASIC classiques 1

CHAPITRE II :

Généralités 5

CHAPITRE III :

Introduction à la programmation en BASIC GfA 7

CHAPITRE IV :

L'éditeur 9

CHAPITRE V :

Les opérateurs 24

ANNEXE A :

Les instructions et fonctions 33

ANNEXE B :

Les messages d'erreurs 280

ANNEXE C :

Le code ASCII 285

ANNEXE D :

Les variables et leur organisation 290

ANNEXE E :

Les fonctions spéciales / Divers 295

AVANT PROPOS

C'est une réalité : le BASIC est le langage de programmation le plus répandu.

Cependant, la réputation du BASIC n'est pas toujours excellente. On peut, par exemple, s'en rendre compte dans les universités et les grandes écoles où l'on enseigne plutôt le langage PASCAL que le langage BASIC (dans certaines écoles, le BASIC est même totalement mis à l'écart).

La principale raison est que le PASCAL permet une programmation structurée. Mais le langage BASIC a évolué et a subi de nombreuses améliorations. Prenons l'exemple de l'instruction *IF ... THEN* : tout au début, l'instruction THEN ne pouvait être suivi que par un numéro de ligne, ce qui entraînait une multitudes d'instructions de saut. Ensuite, on a pu mettre après le THEN une série d'instructions. Actuellement, presque tous les interpréteurs BASIC possèdent l'instruction ELSE.

La démarche suivie pour le développement du BASIC GfA n'était pas simplement d'ajouter de petites améliorations. Le but était plutôt de développer un langage BASIC entièrement nouveau, qui devait répondre aux conditions suivantes :

- Une programmation structurée devait être rendue possible dans tous les domaines.

- Quelqu'un, qui avait déjà programmé en BASIC, devait pouvoir se servir du nouvel interpréteur en peu de temps.

- Les avantages du BASIC devaient impérativement être maintenus dans la nouvelle version.

Avec le BASIC GfA, vous disposez d'un interpréteur BASIC qui non seulement répond aux conditions ci-dessus, mais qui offre encore des avantages supplémentaires :

- L'interpréteur BASIC GfA est très compact. Il n'occupe que 55 Koctets environ de la place mémoire (si précieuse) de votre ATARI ST.

- L'interpréteur possède une vitesse d'exécution exceptionnellement rapide. Ainsi une boucle vide FOR-NEXT comprenant 10000 passages dure moins d'une demi seconde.

- Le BASIC GfA travaille avec une précision de 11 chiffres.

CHAPITRE I.

Avantages par rapport aux interpréteurs BASIC classiques

Les différences fondamentales sont au nombre de trois :

* En BASIC GfA, il n'y a pas de numéros de ligne.

* Chaque ligne de programme ne peut contenir qu'une seule instruction.

* Il existe plusieurs instructions de structure nouvelles pour le BASIC.

Certaines instructions traditionnelles ont une nouvelle syntaxe plus adaptée à la structure du BASIC GfA.

La différence qui saute aux yeux, quand on parcourt le listing d'un programme écrit en BASIC GfA, est l'absence de numérotation des lignes. On est tellement habitué à ce qu'en BASIC, les lignes soient précédées d'un numéro, qu'on ne se pose même plus la question : *à quoi sert donc cette numérotation ? :*

- Grâce aux numéros de ligne, les instructions GOTO, GOSUB mais aussi IF...THEN permettent de se brancher à n'importe quel morceau du programme.

- Quand les lignes sont numérotées de dix en dix, comme c'est le cas couramment, on peut insérer plusieurs lignes entre deux instructions.

- La recherche, l'affichage, la modification ou l'effacement d'une partie du programme est rendue possible.

Les deux derniers points peuvent être réalisés de bien meilleure façon grâce à un bon éditeur, qui fait partie du BASIC GfA. Le premier point peut également être résolu simplement, en employant des étiquettes (label).

De cette manière, la numérotation des lignes devient superflue. De plus, on économise ainsi de la place mémoire.

Comme nous l'avons expliqué dans l'avant-propos, le BASIC GfA autorise une programmation structurée. En outre, de nouvelles instructions ont été ajoutées à celles déjà existantes :

DO...LOOP
WHILE...WEND
REPEAT...UNTIL
PROCEDURE (avec des variables locales)

Pour améliorer la lisibilité d'un programme, il était par ailleurs intéressant de n'autoriser qu'une seule instruction par ligne.

Regardez l'exemple de programme qui suit ; il montre la nouvelle structure de l'instruction IF...THEN et met en évidence la clarté du listing :

```
INPUT A
IF A>0 THEN
 PRINT 'positif'
    . . .
    . . .
ELSE
  IF A=0 THEN
    PRINT 'nul'
      . . .
      . . .
  ELSE
```

```
            PRINT 'négatif'
              . . .
              . . .
          ENDIF
            . . .
            . . .
        ENDIF
```

Les points de suspension indiquent les domaines où l'on peut insérer plusieurs autres instructions, sans pour cela modifier la structure globale.

CHAPITRE II.

Généralités

En plus de l'interpréteur BASIC GfA, il y a sur la disquette un fichier portant le nom 'GFABASRO.PRG'. Il s'agit de la version 'RUN-ONLY' du BASIC GfA. Avec cette version, il est impossible d'écrire de nouveaux programmes, seuls des programmes déjà existants peuvent être chargés et exécutés. Vous êtes autorisé à donner cette version à d'autres personnes. Ainsi, le cercle des utilisateurs du ST qui ne possède pas encore le BASIC GfA, ont la possibilité d'utiliser vos propres programmes écrits en GfA.

Remarque :

Le programme suivant permet de modifier la présentation du RUN-ONLY ou de lancer automatiquement un programme :

```
' Insérez ici une des deux lignes suivantes
'
' Présentation:
' Phrase$=chr$(27)+"HGFA-BASIC Run-Only"
'
' Autostart:
' Phrase$=chr$(0)+"PROGRAM.BAS"
'
' le texte ne peut exéder 60 caractères
Open "U",#1,"GFABASRO.PRG"
Seek #1,30
Print #1,Phrase$
Print #1,chr$(0)
Close #1
```

CHAPITRE III.

Introduction à la programmation en BASIC GfA

Après avoir chargé le BASIC GfA à partir de la disquette, vous vous trouvez dans l'éditeur. Vous pouvez donc immédiatement commencer à programmer. Ayez cependant encore un peu de patience ! Il vous est tout d'abord nécessaire de connaître l'environnement du BASIC GfA :

Vous disposez de deux fenêtre-écrans, que vous pouvez utiliser indépendamment. Une des fenêtres représente l'éditeur, que vous reconnaitrez au menu situé sur la bande supérieure. L'autre fenêtre est un écran de restitution. C'est ici que sera affiché le résultat de l'exécution du programme.

Si vous vous trouvez dans l'éditeur, vous pouvez accéder à l'écran de restitution de trois façons :

* *Exécution du programme en appuyant simultanément sur les touches <SHIFT> et <F10>, ou en choisissant l'option RUN du menu*
(Le retour après la fin de l'exécution se fait en appuyant sur <RETURN>).

* *Emploi de la touche <F9> ou choix de l'option FLIP.*
(Retour à l'éditeur en appuyant sur une touche quelconque du clavier ou sur le bouton gauche de la souris).

* *Emploi des touches <ESC> ou <SHIFT> et <F9>, ou choix de l'option DIRECT avec la souris.*

Ce faisant, vous vous placez en Mode Direct : les instructions entrées au clavier sont directement exécutées sans être insérées au programme.

Pour pouvoir lire le résultat sur le moniteur vidéo, le mode direct utilise l'écran de restitution (on quitte le mode direct en appuyant sur <ESC> puis sur <RETURN>, ou en tapant l'ordre EDIT (abrév. : ED), ou encore par la fonction STOP (<SHIFT>+<ALTERNATE>+<CONTROL>))

Voyons cela sur un exemple :

Vous vous trouvez dans l'éditeur. Entrez les instructions suivantes :

```
BOX 140,100,500,300
PCIRCLE 320,200,100
```

Exécutez maintenant ce programme comme cela est indiqué ci-dessus. Pour revenir à l'éditeur, vous pouvez appuyer sur <RETURN> ou choisir l'option <RETURN> avec la souris.

En appuyant plusieurs fois de suite sur la touche F9, vous pouvez passer d'un écran à l'autre.

Passez maintenant en mode direct par l'emploi de <ESC> ou <SHIFT> et <F9> et tapez :

```
PRINT AT(36,12);'BASIC GfA'
```

Le résultat est immédiatement affiché dès que la touche <RETURN> est enfoncée.

Vous pouvez quitter le mode direct grâce aux touches <ESC> puis <RETURN>, ou en utilisant l'instruction EDIT. Les deux fenêtres doivent maintenant vous être familières et nous pouvons nous consacrer à l'étude détaillée des possibilités de l'éditeur.

CHAPITRE IV.

L'éditeur

GENERALITES.

L'éditeur qui a été réalisé spécialement pour le développement de programmes est orienté vers l'utilisation des fenêtres, en particulier pour les opérations de blocs et la recherche ou la substitution de parties de programmes.

Sur le bord supérieur de l'écran se trouvent deux lignes de menu réservées à la description des touches-fonctions ou au choix des différentes options avec la souris. Le reste de l'écran est entièrement disponible pour le programme.

On peut choisir deux tailles différentes de texte en utilisant les touches <SHIFT> et <F8> ou en choisissant respectivement l'option TEXT 8 ou TEXT 16 avec la souris (uniquement avec un moniteur vidéo monochrome). Avec la taille normale (TEXT 16), il y a 23 lignes de programme ; avec la police de caractères plus petite (TEXT 8), on dispose de 48 lignes. Ce dernier mode permet d'éditer et de lister les programmes de manière très lisible.

Chaque ligne de programme ne contient qu'une seule instruction BASIC. Une ligne de programme contient au maximum 255 caractères ou 255 octets. Comme par exemple un nombre réel à virgule occupe 6 octets en mémoire (et parfois même 8 octets), le message d'erreur *'LIGNE TROP LONGUE'* peut apparaître bien avant la saisie de 255 caractères.

Comme une ligne d'écran ne peut contenir que 80 caractères, le début de la ligne disparait vers la gauche de l'écran lors d'une saisie de plus de 80 caractères. C'est uniquement la ligne affectée qui subit ce déplacement.

Le restant du programme n'est pas modifié sur l'écran. La saisie de la ligne de programme une fois terminée, après avoir appuyé sur <RETURN> par exemple, seuls les 79 premiers caractères de la ligne de programme sont représentés, éventuellement suivis par le symbole *'FLECHE VERS LA DROITE'* qui indique que la ligne contient plus de 80 caractères et n'apparait donc pas dans sa totalité sur l'écran.

La saisie de la ligne de programme s'effectue à partir du clavier. Les caractères spéciaux, qui n'ont pas de touche réservée, peuvent également être édités (voir annexe C).

La fin d'une ligne est obtenue en appuyant sur <RETURN> ou sur une des commandes de déplacement du curseur qui entraine l'abandon de la ligne actuelle (voir plus bas).

A ce moment là, la syntaxe de la ligne est contrôlée et, le cas échéant, le message d'erreur *'ERREUR DE SYNTAXE'* apparait. La structure du programme est mise en évidence par décalage des lignes d'instruction (indentation) et les instructions éventuellement écrites en abrégé sont réécrites en entier (voir annexe A).

Les différentes commandes de l'éditeur peuvent être classées en cinq catégories :

- Commandes de déplacement du curseur
- Commandes de modification du programme
- Commandes de recherche et de substitution
- Commandes de blocs
- Autres commandes de menu

Les différentes commandes vont maintenant être décrites de façon détaillée.

COMMANDES DE DEPLACEMENT DU CURSEUR.

<FLECHE VERS LA GAUCHE>

Déplace le curseur à l'intérieur d'une ligne d'un caractère vers la gauche.

<FLECHE VERS LA DROITE>

Déplace le curseur à l'intérieur d'une ligne d'un caractère vers la droite.

<CONTROL>+<FLECHE VERS LA DROITE>

Positionne le curseur en fin de ligne.

<CONTROL>+<FLECHE VERS LA GAUCHE>

Positionne le curseur en début de ligne.

<TAB>

Déplace le curseur d'une tabulation vers la droite. Les différentes positions de la tabulation sont des multiples de 8 et sont indiquées par les traits verticaux dans la bande-menu pour les lignes de moins de 80 caractères.

<CONTROL>+<TAB>

Déplace le curseur d'une tabulation vers la gauche.

<FLECHE VERS LE BAS> ou <RETURN>

Place le curseur au début de la ligne suivante.

<FLECHE VERS LE HAUT>

Place le curseur au début de la ligne précédente.

<CONTROL>+<FLECHE VERS LE BAS> ou <F7>

Déplace le programme d'une page-écran vers le bas et place le curseur au début de cette page (cette commande peut aussi être obtenue en choisissant l'option 'PG DOWN' du menu).

<CONTROL>+<FLECHE VERS LE HAUT> ou <SHIFT>+<F7>

Déplace le programme d'une page-écran vers le haut et place le curseur au début de cette nouvelle page (cette commande peut aussi être obtenue en choisissant l'option 'PG UP' du menu).

<HOME>

Positionne le curseur au début de la page-écran en train d'être traitée. Le listing est alors automatiquement formaté (indentation, c'est-à-dire décalage des instructions).

<CONTROL>+<HOME>

Place le curseur au début du programme.

<CONTROL>+<Z>

Place le curseur en fin de programme. Il est également possible de déplacer le curseur à l'aide de la souris. En choisissant une nouvelle position avec la souris et en la validant, on déplace le curseur à l'endroit de la souris. Si, ce faisant, on quitte la ligne courante, il s'en suit naturellement une vérification de la syntaxe.

COMMANDES DE MODIFICATION DU PROGRAMME.

<F8>

En appuyant sur cette touche (ou en choisissant l'option INSERT ou OVERWRT avec la souris), on passe d'un mode d'écriture à un autre : le mode Insert (mode insertion) et le mode Overwrite (mode surimpression).

Le mode insertion, dans lequel on se trouve lors de l'appel de l'éditeur, vous permet d'insérer du texte dans une ligne déjà existante. Les caractères de droite sont décalés d'autant vers la droite.

Dans le mode surimpression, le caractère sous le curseur est remplacé par le nouveau caractère saisi au clavier.

<BACKSPACE>

Efface le caractère situé à gauche du curseur. Le curseur, le caractère sous le curseur et tous ceux de droite sont décalés d'un cran vers la gauche.

<DELETE>

Efface le caractère sous le curseur. Tous les caractères de droite sont décalés d'un cran vers la gauche.

<INSERT>

Déplace la ligne, au début de laquelle se trouve le curseur, et toutes celles qui suivent, d'une ligne vers le bas. Le curseur lui-même ne bouge pas et se trouve donc au début d'une ligne vide, prêt pour écrire une nouvelle ligne de programme. La nouvelle ligne une fois entrée et la touche <RETURN> enfoncée, ce processus se répète à nouveau automatiquement, jusqu'à ce que la touche <RETURN> soit enfoncée en début d'une ligne vide ou jusqu'à ce que la ligne courante soit quittée par une commande de déplacement du curseur.

<UNDO>

Effectue toutes les modifications à l'intérieure d'une ligne par retour en arrière et remet la ligne dans l'état dans lequel elle se trouvait lorsque le curseur a atteint la ligne par une commande de déplacement du curseur. Cette commande n'est effective que si la syntaxe de la ligne n'a pas encore été vérifiée.

Par ailleurs, la page-écran est réécrite et le listing est formaté (indentation), comme pour la touche <HOME>.

Le mode insertion de ligne, s'il a été mis en place grâce à la touche <INSERT> par exemple, est déconnecté.

En mode direct, UNDO permet d'obtenir la dernière ligne entrée.

<CONTROL>+<DELETE>

Efface de façon irréversible la ligne où se trouve le curseur et déplace simultanément toutes les lignes suivantes d'une ligne vers le haut. Le curseur reste sur la même ligne et se trouve, après exécution de la commande, au début de la ligne qui suivait celle qui vient d'être effacée. Cette commande ne peut pas être rattrapée par <UNDO> !

COMMANDES DE RECHERCHE ET DE SUBSTITUTION.

<F6> (= FIND du menu)

La première ligne du menu est effacée et il apparait 'FIND :' Vous pouvez y entrer une chaîne de caractères à rechercher faisant au maximum 60 caractères de long. La chaîne entrée peut être corrigée par <DELETE> et <BACKSPACE>. Avec les touches de déplacement du curseur 'DROITE' et 'GAUCHE', on peut atteindre pas à pas n'importe quel caractère de la partie de la chaîne déjà écrite. Avec les touches 'VERS LE HAUT' et 'VERS LE BAS', on positionne le curseur respectivement à la fin et au début de la chaîne de caractères.

Après validation de la chaîne recherchée par <RETURN>, le curseur se positionne au début d'une ligne contenant cette chaîne ou en fin de programme si la chaine n'est pas trouvée.

La recherche commence toujours à partir de la ligne situé en dessous du curseur de programme. Il est donc astucieux de placer le curseur en début de programme par <CONTROL> +<HOME> avant d'appeler la commande FIND.

La recherche des autres lignes dans lesquelles se trouvent la chaîne de caractères recherchée est possible par <CONTROL> +<F> (voir plus loin).

Si l'on s'est trompé, on peut sortir de la commande FIND par <ESC> suivi de <RETURN>.

<SHIFT>+<F6> (= REPLACE du menu)

La première ligne du menu est effacée et il apparait 'FIND :'. Puis, on entre la chaîne de caractères comme pour la commande 'FIND'.

Ensuite, la deuxième ligne du menu est également effacée et il apparait 'REPLACE :'. Ici aussi, on peut entrer une chaîne ayant au maximum 60 caractères avec les mêmes possibilités de correction que pour 'FIND'. La substitution de la première chaîne de caractères par la seconde a lieu lorsque les touches <CONTROL> + <R> sont enfoncées (voir plus loin). A part cela, le processus est le même que pour 'FIND'.

Il faut encore noter que la commande 'FIND' aussi bien que 'REPLACE' tiennent compte des minuscules et des majuscules. Cela signifie qu'une chaîne n'est trouvée que si elle correspond exactement à la chaîne recherchée.

<CONTROL>+<F>

Recherche la chaîne de caractères qui a été entrée lors de la commande 'FIND' ou 'REPLACE'.

La chaîne recherchée reste en mémoire aussi longtemps qu'une nouvelle recherche n'est pas effectuée.

La recherche commence à partir de la ligne en dessous du curseur, elle n'est donc pas effectuée dans la ligne où se trouve le curseur. Après le processus de recherche, le curseur se trouve au début d'une ligne contenant la chaîne recherchée, ou bien, si la chaîne n'a pas été rencontrée, il se positionne en fin de programme.

<CONTROL>+<R>

Recherche la chaîne de caractères enregistrée par la commande 'REPLACE', ou remplace la chaîne recherchée par la chaîne à substituer.

Si cette commande est effectuée dans une ligne où se trouve la chaîne recherchée, la substitution a lieu (la substitution peut naturellement être rattrapée grâce à la commande UNDO, aussi longtemps que le curseur se trouve sur la ligne en question).

Si cette commande est effectuée dans une ligne ne contenant pas la chaîne recherchée, elle a le même effet que la commande <CONTROL>+<F>, ce qui signifie qu'on recherche la prochaine ligne contenant la chaîne de caractères voulue (si l'on ne souhaite pas effectuer de substitution dans une ligne où apparait la chaîne recherchée, il suffit de poursuivre la recherche par <CONTROL>+<F>).

COMMANDES DE BLOCS.

<SHIFT>+<F5> (= BLK STA du menu)

Repère le début d'un bloc.

Un bloc est une partie du programme et est composé de lignes de programme complètes et successives.

Si la commande BLK STA (Block Start) est choisie, le début du bloc correspondra au début de la ligne où se trouvait le curseur avant d'effectuer cette commande.

<F5> (= BLK END du menu)

Repère la fin d'un bloc.

La fin du bloc correspond au début de la ligne où se trouvait le curseur. Cela signifie que la ligne du curseur ne fait pas partie du bloc.

Un bloc n'est défini que si le début et la fin du bloc ont été repérés. Le bloc est alors mis en évidence par un affichage différent.

Un bloc et sa représentation peuvent être effacés en faisant coïncider sur la même ligne le début et la fin du bloc.

<F4> (= BLOCK du menu)

Choix de l'opération de bloc.

Si l'un au moins des repères manque, il apparait le message d'erreur 'BLOC ???' (pour continuer, appuyer sur la souris), sinon, la première ligne de menu est remplacée par un autre menu comprenant plusieurs choix possibles qui seront étudiés en détail un peu plus loin.

Si la commande a été choisie par erreur, on peut revenir en arrière par action sur la souris.

<C> (= COPY)

Le bloc défini est recopié avant la ligne où se trouvait le curseur avant le choix de la commande 'BLOCK'.

Ceci n'est naturellement possible que si le curseur se trouve en dehors du bloc. Sinon, le message d'erreur *'CURSOR IN BLOCK"* (curseur dans le bloc) apparait (pour continuer, appuyer sur la souris).

<M> (= MOVE)

Déplace le bloc avant la ligne où se trouvait le curseur avant le choix de la commande 'BLOCK'.

Comme pour la commande 'COPY', cela n'est possible que si le curseur se trouvait en dehors du bloc. Sinon, le message d'erreur *'CURSOR IN BLOCK'* (Curseur dans le bloc) apparait (pour continuer, appuyer sur la souris).

(En réalité, un 'MOVE' s'effectue en deux temps : un 'COPY', puis un 'DELETE').

<W> (= WRITE)

Sauvegarde le bloc sur disquette dans un fichier TEXT (.LST).

<L> (= LLIST)

Imprime le bloc sur imprimante.

<S> (= START)

Positionne le curseur en début de bloc.

<E> (= END)

Positionne le curseur en fin de bloc.

<CONTROL>+<D> (= DEL)

Efface le bloc de façon irréversible.

<H> (= HIDE)

Efface les repères de bloc.

AUTRES COMMANDES DU MENU.

Le choix de n'importe quelle option du menu peut être effectué aussi bien par la touche-fonction correspondante que par validation avec la souris.

<F1> (= LOAD)

Charge un programme à partir de la disquette.

Le suffixe *.BAS est utilisé par défaut.

Tout programme qui se trouverait éventuellement dans l'éditeur est effacé sauf si l'on clique dans 'Annuler' du sélecteur d'objet.

<SHIFT>+<F1> (= SAVE)

Sauvegarde le programme actuellement dans l'éditeur sur disquette.

Si aucune extension n'a été précisée, le suffixe .BAS est automatiquement choisi.

<F2> (= MERGE)

Charge un fichier TEXT à partir de la disquette (*.LST par défaut) et insère les lignes correspondantes dans le programme avant la ligne où se trouve le curseur.

S'il y a erreur de syntaxe, le signe ==> apparait au début de la ligne incriminée.

<SHIFT>+<F2> (= SAVE,A)

Sauvegarde le programme contenu dans l'éditeur sur disquette.

Le suffixe .LST est pris par défaut.

<F3> (=LLIST)

Sort tout le programme sur imprimante.

L'impression du listing peut être interrompue en appuyant sur les touches <Alternate>+<Control>+<Shift>. Si l'imprimante n'est pas connectée ou <Off-line>, l'éditeur est à nouveau disponible après 30 secondes environ.

<SHIFT>+<F3> (= QUIT)

Permet de quitter l'interpréteur et de revenir au Bureau du GEM.

<SHIFT>+<F4> (= NEW)

Efface le programme contenu en mémoire.

<F9> (= FLIP)

Passe sur la fenêtre de restitution (voir l'introduction).

<F10> (= TEST)

Teste si toutes les boucles sont refermées dans l'ordre.

<SHIFT>+<F10> (= RUN)

Exécute le programme.

RESUME DES COMMANDES DE L'EDITEUR.

(^ signifie CONTROL)

FLECHE GAUCHE	: Curseur d'un cran à gauche
FLECHE DROITE	: Curseur d'un cran à droite
^FLECHE GAUCHE	: Curseur en début de ligne
^FLECHE DROITE	: Curseur en fin de ligne
TAB	: Curseur sur tabulation de droite
^TAB	: Curseur sur tabulation de gauche
FLECHE EN BAS	: Curseur d'une ligne vers le bas
RETURN	: Curseur d'une ligne vers le bas
FLECHE EN HAUT	: Curseur d'une ligne vers le haut
^FLECHE EN BAS	: Page-écran suivante
PG DOWN	: Page-écran suivante
^FLECHE EN HAUT	: Page-écran précédente
PG UP	: Page-écran précédente
HOME	: Curseur en début de page
^HOME	: Curseur en début de programme
^Z	: Curseur en fin de programme
F8	: Insertion/Surimpression
BACKSPACE	: Retour du curseur avec effacement
DELETE	: Effacement du caract. sous curseur
INSERT	: Insertion d'une ligne
UNDO	: Retour à l'état précédent
^DEL	: Effacement de la ligne
FIND	: Saisie de la chaîne recherchée
REPLACE	: idem + saisie de la nouvelle chaîne
^F	: Recherche d'une chaîne
^R	: Substitution d'une chaîne
BLK STA	: Marquage du début du bloc
BLK END	: Marquage de la fin du bloc
BLOCK	: Choix de l'opération de bloc :
C	: Copie de bloc
M	: Déplacement de bloc
W	: Sauvegarde de bloc
L	: Impression de bloc

S	: Curseur en début de bloc
E	: Curseur en fin de bloc
^D	: Effacement de bloc
H	: Effacement des marques de bloc
LOAD	: Chargement du programme
SAVE	: Sauvegarde du programme
MERGE	: Insertion d'un fichier TEXT
SAVE,A	: Sauvegarde sur fichier TEXT
LLIST	: Sortie sur imprimante
QUIT	: Retour au Desktop
NEW	: Effacement du programme
FLIP	: Passage sur fenêtre de restitution
DIRECT	: Mise en place du mode direct
TEST	: Test de validité des boucles
RUN	: Exécution du programme

CHAPITRE V.

Les opérateurs

On distingue les opérations numériques et les opérations entre chaînes de caractères, selon que le résultat est un nombre ou une chaîne de caractères (string).

Il y a deux opérateurs de chaînes de caractères :

* *La concaténation de chaînes de caractères par l'opérateur + :*

Les chaînes de caractères sont mises à la suite les unes des autres pour n'en former plus qu'une.

Exemple : Si A$='BASIC', B$='-' et C$='GfA', la concaténation des 3 chaînes donne : A$+B$+C$ = 'BASIC-GfA'.

* *Les fonctions sur les chaînes de caractères :*

LEFT$, RIGHT$, MID$, SPACE$, STRING$ et STR$.
Ces fonctions seront décrites en détail dans l'annexe A.

Il y a 4 types d'opérateurs numériques :

1. Les opérateurs arithmétiques
2. Les opérateurs de comparaison
3. Les opérateurs logiques
4. Les fonctions numériques

Les fonctions numériques sont examinées de façon détaillée dans l'annexe A.

1. LES OPERATEURS ARITHMETIQUES.

Le BASIC GfA connaît les opérations arithmétiques suivantes (opérations de calcul) :

^ Puissance.
Par exemple : 2^3=2*2*2=8

- Signe négatif.

* Multiplication.

/ Division.

DIV Division entière.
(\) On prend la partie entière du résultat de la division normale.
On peut donc écrire :
A DIV B = A\B = TRUNC(A/B).
Ex : 3.7 DIV 0.7 = 5
-17 DIV -4 = 4

MOD Modulo. Le modulo correspond au reste de la division entière.
Ainsi :
A MOD B = FRAC(A/B)*B
Ex : 3.7 MOD 0.7 = 0.2
-17 MOD -4 = -1

+ Addition

- Soustraction

2. LES OPERATEURS DE COMPARAISON.

Les opérateurs de comparaison s'appliquent à deux expressions numériques ou à deux chaines de caractères.

Suivant la valeur booléenne prise par le résultat (vrai ou faux), on attribut à ce résultat la valeur -1 (vrai) ou 0 (faux) (comme ce résultat est dans tous les cas un nombre, la comparaison de deux chaînes de caractères est aussi considérée comme une opération numérique).

On dispose des opérateurs de comparaison suivants :

=	égal.
>	strictement supérieur.
<	strictement inférieur.
<>	différent de.
>= ou =>	supérieur ou égal.
<= ou =<	inférieur ou égal.

Voici quelques exemples de comparaison d'expressions numériques :

PRINT 3=1.5*2	donne la valeur -1 (vrai)
PRINT 5>5	donne la valeur 0 (faux)
PRINT 7<>17	donne la valeur -1 (vrai)
PRINT 8<=8	donne la valeur -1 (vrai)

La comparaison entre deux chaînes de caractères obéit aux règles suivantes :

* Deux chaînes de caractères sont égales, si elles sont rigoureusement identiques (tous les caractères sont pris en compte, même les blancs et la ponctuation).

Ainsi par exemple : ' I , 7?:'=' I , 7?:'

* Pour la comparaison de deux chaînes de caractères, l'interpréteur procède de la manière suivante :

L'interpréteur compare les valeurs des codes ASCII (voir annexe C) des deux premiers caractères des deux chaînes. Si ces valeurs sont différentes, alors la chaîne avec le code ASCII le plus élevé est 'supérieure' à l'autre.

Si les valeurs sont identiques, on compare les deux codes ASCII des deux caractères suivants, et ainsi de suite jusqu'à ce que ou bien les deux caractères sont différents, ou bien la fin d'une des deux chaînes a été atteinte. Dans ce dernier cas, la chaîne la plus longue est la plus 'grande'.

Voici quelques exemples illustrants ce propos :

```
PRINT ('GfA' > 'GfA ')      donne  0 (faux)
PRINT ('123' < 'ABC')       donne -1 (vrai)
PRINT ('abc' < 'ABC')       donne  0
PRINT ('May' > 'Mai')       donne -1
PRINT (' Z'  > 'A ')        donne  0
```

3. LES OPERATEURS LOGIQUES.

Les opérateurs logiques permettent de relier entre elles des expressions ou des relations entre expressions.

En règle générale, les opérateurs logiques s'appliquent à des variables booléennes (une variable booléenne peut prendre deux valeurs : Vrai ou Faux) et donne un résultat également booléen. De plus, on attribue à toute expression numérique différente de 0 la valeur Vrai et à toute expression numérique égale à 0 la valeur Faux.

Pour y voir plus clair, nous allons maintenant examiner les différents opérateurs logiques un par un. Par la suite, v signifiera vrai et f sera l'abréviation de faux.

AND (ET logique)

Le résultat d'un AND est seulement v, si tous les deux arguments sont v :

A	B	A AND B
v	v	v
v	f	f
f	v	f
f	f	f

Exemples :

```
PRINT 3=3 AND 4>2   donne -1 (v)
PRINT 3>3 AND 5>3   donne  0 (f)
```

OR (OU logique (non exclusif))

Le résultat d'un OR est seulement f, si tous les deux arguments sont f :

A	B	A OR B
v	v	v
v	f	v
f	v	v
f	f	f

Exemples :

```
PRINT 'A'<'a'  OR 3=5     donne -1 (v)
PRINT  2>7     OR 10=20   donne  0 (f)
```

XOR (OU exclusif)

Le résultat d'un XOR est f, si les deux arguments ont la même valeur :

A	B	A XOR B
v	v	f
v	f	v
f	v	v
f	f	f

Exemples :

```
PRINT 3=3 XOR 6=6      donne  0 (f)
PRINT 3>5 XOR 3<5      donne -1 (v)
```

NOT (Négation)

Elle inverse la valeur de la variable booléenne.

A	NOT A
v	f
f	v

Exemple :

```
PRINT NOT 5>7          donne -1 (v)
```

IMP (Implication)

L'implication est seulement **f**, lorsque quelque chose de vrai entraine quelque chose de faux.

A	B	A IMP B
v	v	v
v	f	**f**
f	v	v
f	f	v

Exemples :

```
PRINT 3=5 IMP 2>3       donne -1 (v)
PRINT 4>2 IMP 1=1*2     donne  0 (f)
PRINT 3<4 IMP 5=5       donne -1 (v)
```

EQV (Equivalence)

L'équivalence est identique à (A IMP B) AND (B IMP A).

A	B	A EQV B
v	v	v
v	f	**f**
f	v	f
f	f	v

Exemples :

```
PRINT 'A '<'AB' EQV 3=3     donne -1 (v)
PRINT  3+3>8  EQV 2+2=4     donne  0 (f)
```

Les opérateurs logiques sont principalement utilisés pour permettre différents déroulements possibles d'un programme, selon la valeur booléenne (Vrai ou Faux) liée à la comparaison de deux expressions (grâce aux instructions IF...THEN...ELSE et EXIT IF).

Les opérateurs logiques peuvent aussi servir à tester une série de bits, comme nous allons le voir maintenant :

Les opérandes auxquels peuvent s'appliquer des opérateurs logiques doivent être compris entre -2147483648 et +2147483647, car les opérateurs logiques travaillent avec des mots de 32 bits (pour la représentation des nombres négatifs, on utilise la complémentation à deux).

Les opérandes sont réduits à des nombres entiers et traités bit par bit. Chacun des 32 bits du résultat est obtenu en appliquant l'un des opérateurs décrits ci-dessus aux bits correspondants des opérandes. 1 signifie ici v (vrai) et 0 f (faux).

Cela va se clarifier avec l'exemple qui suit :

PRINT 29 AND 267 a pour résultat 9

```
00000000 00000000 00000000  00011101 =  29
00000000 00000000 00000001  00001011 = 267
-----------------------------------------
00000000 00000000 00000000  00001001 =   9
```

PRINT -8 EQV 30 a pour résultat 25

```
11111111 11111111 11111111  11111000 =  -8
00000000 00000000 00000000  00011110 =  30
-----------------------------------------
00000000 00000000 00000000  00011001 =  25
```

PRIORITE DES OPERATEURS.

Les opérateurs sont traités dans l'ordre suivant :

( )	Les parenthèses sont prioritaires sur tout le reste. Elles permettent de réaliser n'importe quelle opération.
^	Puissance.
+, -	Signes.
*, /	Multiplication et division.
DIV, \, MOD	Division entière et modulo.
+, -	Addition et soustraction.
=,<>,<,>,<=,>=	Opérations de comparaison.
NOT, AND, OR, XOR, IMP, EQV	Opérations logiques.

ANNEXE A.

Instructions et fonctions

Cette annexe contient la description de toutes les instructions et fonctions disponibles, dans l'ordre alphabétique.

La différence entre instruction et ordre, qui avait un sens historiquement, est désormais levée (il était d'usage d'appeler ordre ce qui concernait le mode direct et qui contribuait à l'élaboration des programmes. Les instructions, par contre, s'utilisaient à l'intérieur d'un programme. Comme toutes les instructions peuvent être employées aussi bien en mode direct que dans un programme, cette distinction n'est plus nécessaire).

La description des instructions s'organise en cinq parties, celle des fonctions en quatre :

1. Syntaxe :

Ici est décrit le format de l'instruction ou de la fonction.

Les mots écrits en majuscules sont des mots-clés.

Les expressions entre crochets sont optionnelles, c'est-à-dire facultatives. Trois petits points '...' indiquent que l'expression qui se trouve avant peut être répétée autant de fois que l'on veut.

Les expressions qui interviennent dans le format de l'instruction sont expliquées dans la suite de la description.

2. Abrév. :

Presque toutes les instructions ont une abréviation. On donne ici l'abréviation minimale.

Ainsi, plutôt que de taper l'instruction PRINT en entier, il suffit d'entrer un P. L'éditeur comprend naturellement aussi PR, PRI, PRIN et bien sûr PRINT. Après avoir effectué le contrôle syntaxique, l'éditeur remplace les abréviations éventuelles par les instructions entières correspondantes.

Cette partie est inutile pour les fonctions, car le nom d'une fonction doit toujours être entré dans sa totalité.

3. Exemple :

Dans cette partie, on présente un exemple simple de programme.

D'une part, l'exemple doit mettre en évidence les différentes syntaxes possibles de l'instruction.

D'autre part, l'exemple sert à concrétiser l'utilisation de l'instruction ou de la fonction.

Si, après l'explication de l'instruction ou de la fonction, des doutes subsistent quant à son emploi ou son utilité, il est conseillé de taper et de tester soi-même l'exemple fourni.

Comme certains exemples utilisent des boucles sans fin, il est important de savoir que l'on peut arrêter ou interrompre l'exécution d'un programme par l'action simultanée sur les trois touches <CONTROL>, <SHIFT> et <ALTERNATE>.

4. Fonction :

Décrit en peu de mots quelle est l'utilisation de l'instruction ou de la fonction (c'est-à-dire ce que fait l'instruction ou la fonction).

5. Explication :

Dans cette partie, on explique de façon détaillée ce qu'il est important de connaître pour la mise en application de l'instruction ou de la fonction.

Si on peut préciser le domaine d'application de l'instruction,

Instruction I/O (relative aux entrées/sorties)
Instruction graphique
Instruction arithmétique
Instruction de structure

ce dernier sera précisé dans l'en-tête de la description.

Dans la description concernant les instructions graphiques, nous supposons que l'ordinateur est connecté à un moniteur-vidéo monochrome de haute résolution. Celui-ci peut représenter en deux couleurs (noir & blanc) 640 x 400 pixels.

Il est à noter que pour des résolutions graphiques moyennes, le nombre de pixels est seulement de 640 x 200 (en quatre couleurs), et pour des résolutions encore plus petites, il n'est que de 320 x 200. Dans ce dernier cas, on dispose de 16 couleurs.

LISTE DES INSTRUCTIONS ET DES FONCTIONS (Annexe A)

ABS
ADD
ALERT
ARRAYFILL
ARRPTR
ASC
ATN
BASEPAGE
BGET
BIN$
DIV
DO...LOOP
DRAW
EDIT
ELLIPSE
END
EOF
ERASE

BITBLT
BLOAD, BSAVE
BMOVE
BOX
BPUT

C:
CALL
CHAIN

CHDIR
CHDRIVE
CHR$
CIRCLE
CLEAR
CLEARW
CLOSE
CLOSEW
CLR

CLS
COLOR
CONT
COS
CLSCOL, CLSLIN
CVI, CVL, CVS, CVF, CVD
DATA
DATE$
DEC
DEFNUM
DEFFILL
DEFFN

DEFLINE
DEFLIST
DEFMARK
DEFMOUSE
ERR

ERROR
EVEN
EXEC
EXIST
EXIT
EXP
FALSE
FATAL
FIELD
FILES
FILESELECT
FILL
FIX
FOR...NEXT

FORM INPUT
FORM INPUT AS
FRAC
FRE
FULLW

GET
GET (Rel. Dat.)
GOSUB
GOTO

GRAPHMODE
HARDCOPY
HEX$
HIDEM
HIMEM
IF
INC
INFOW
INKEY$

DEFTEXT

DFREE
DIM
DIM?
DIR
DIR$

KILL
LEFT$
LEN
LET
LINE
LINE INPUT
LINE INPUT #
LIST
LLIST
LOAD
LOC
LOCAL
LOF
LOG, LOG10
LPOS

LPRINT
LSET
MAX
MENU
MID$
MIN
MKDIR
MKI$,MKL$,MKS$,MKF$,MKD$
MONITOR
MOUSE
MOUSEX, MOUSEY, MOUSEK
MUL
NAME

INP
INP?
INPUT
INPUT #
INPUT$
INSTR
INT

POLYLINE,-FILL,-MARK
POS
PRINT
PRINT #
PRINT (#) USING
PROCEDURE
PUT
PUT (rel. Dat.)
QUIT
RANDOM
RBOX
READ
RELSEEK
REM
REPEAT...UNTIL
RESERVE
RESTORE
RESUME
RETURN
RIGHT$
RMDIR
RND
RSET

RUN
SAVE, PASVE
SEEK
SETCOLOR
SETTIME

NEW

OCT$
ODD
ON...GOSUB
ON BREAK (CONT, GOSUB)
ON ERROR, ON ERROR GOSUB
ON MENU BUTTON
ON MENU GOSUB
OPEN

OPENW
OPTION
OPTION BASE 0/1
OUT
OUT?
PAUSE
P(R)BOX,PCIRCLE,PELLIPSE
PEEK, DPEEK, LPEEK
PI
PLOT
POINT
POKE, DPOKE, LPOKE

TEXT
TIME$
TIMER
TITLEW

TROFF
TRON
TRUE
TRUNC
TYPE

SGET
SGN
SHOWM

SIN
SOUND
SPACE$

SPC
SPOKE,SDPOKE,SLPOKE
SPRITE
SPUT

SQR

STOP
STR$
STRING$
SUB
SWAP
SYSTEM
TAB
TAN

UPPER$
VAL
VAL?
VARPTR
VDIBASE
VOID
VSYNC
WAVE
WHILE...WEND
WRITE (#)
*
==

LISTE DES INSTRUCTIONS ET FONCTIONS (Annexe E)

ADDRIN
ADDOUT
BIOS
CONTRL
GB
GCONTRL
GEMDOS
GEMSYS
GINTIN
GINTOUT
INTIN
INTOUT
PTSIN
PTSOUT
VDISYS
WINDTAB
XBIOS

FONCTION ABS

Syntaxe : ABS(x)

Exemples :
```
A=-2329
PRINT ABS(A)
PRINT ABS(&X1010)
PRINT ABS(3*(-7))
```

Fonction : Retourne la valeur absolue de x.

Explication :

x est une expression numérique quelconque.
ABS(x) est la fonction mathématique valeur absolue. La valeur absolue d'une expression numérique est toujours supérieure ou égale à 0. Elle vaut :

- si x>0, alors ABS(x)=x
- si x=0, alors ABS(x)=0
- si x<0, alors ABS(x)=-x

INSTRUCTION ARITHMETIQUE ADD

Syntaxe : ADD var,n

Abrev. : AD

Exemple :

```
T=TIMER
FOR I%=1 TO 10000
  ADD A%,5
NEXT I%
PRINT (TIMER-T)/200
A%=0
T=TIMER
FOR I%=1 TO 10000
  A%=A%+5
NEXT I%
PRINT (TIMER-T)/200
```

Fonction : Additionne n à la valeur de var.

Explication :

'*var*' doit être une variable numérique ou un élément d'un tableau numérique. 'n' est une expression numérique.

ADD var,n est identique à var=var+n. L'avantage de l'instruction ADD réside dans la vitesse d'exécution (testez l'exemple !). Ainsi, l'utilisation de l'instruction ADD permet d'augmenter sensiblement la vitesse d'exécution, qui par ailleurs est déjà très grande (dans notre exemple, elle varie presque du simple au double).

INSTRUCTION GRAPHIQUE ALERT

Syntaxe : ALERT a,mstring,b,bstring,var

Abrev. : A

Exemple :
```
M$="Alors,|Comment ça va?"
Alert 2,M$,1,"bien|mal",B
PRINT B
```

Fonction : Met en place une boîte d'alarme au centre de l'écran.

Explication :

Avec une boîte d'alarme, on peut prévenir l'utilisateur, le questionner ou lui donner des indications.

'a' permet de choisir le symbole qui sera représenté à l'intérieur de la fenêtre d'alarme : 0=aucun, 1=!, 2=?, 3=STOP.

'mstring' est une expression alphanumérique qui contient le texte principal. On peut représenter au maximum 4 lignes de texte de 30 caractères chacune. Les lignes sont séparées dans la chaîne de caractères par le caractère "|" . Le cas échéant, les lignes sont tronquées ou ignorées.

'bstring' est une expression alphanumérique qui contient le texte relatif à chaque bouton (3 au maximum). Les messages sont comme précédemment séparés par le caractère !. Chaque message est limité à 8 caractères. La variable b (0,1,2,3) indique quel bouton sera représenté plus intensément. Ce bouton peut ensuite être valider par la touche <RETURN>. Enfin, var contient le numéro du bouton qui a été actionné. Pour une meilleure compréhension de cette instruction, il est fortement conseillé d'entrer au clavier l'exemple ci-dessus.

INSTRUCTION ARRAYFILL

Syntaxe : ARRAYFILL tabl(),n

Abrev. : AR

Exemples :
```
DIM A%(4,5,6),B(100)
C=7
ARRAYFILL A%(),12
ARRAYFILL B(),C
PRINT A%(0,0,0),A%(4,5,6),B(80)
```

Fonction : Remplit un tableau avec la valeur n.

Explication :

'tabl( )' est un tableau numérique ou un tableau de booléens quelconque. Cette instruction affecte la valeur n à tous les éléments de tabl().

'n' peut être un nombre ou une variable numérique.

Ainsi, l'instruction tirée de l'exemple :

```
ARRAYFILL A%(),12
```

est identique au bloc d'instructions :

```
FOR I=0 TO 4
  FOR J=0 TO 5
    FOR K=0 TO 6
      A%(I,J,K)=12
    NEXT K
  NEXT J
NEXT I
```

FONCTION **ARRPTR**

Syntaxe : ARRPTR(var)

Exemple :
```
A$="BASIC"
PRINT ARRPTR(A$)
PRINT LPEEK(ARRPTR(A$))
PRINT DPEEK(ARRPTR(A$)+4)
```

Fonction : Détermine l'adresse du descripteur d'une chaîne de caractères ou d'un tableau

Explication :

'var' est une variable alphanumérique (chaîne de caractères) ou un tableau.

Les chaînes de caractères et les tableaux sont placés en mémoire suivant un schéma précis à l'aide d'un descripteur (voir annexe D : organisation des variables).

ARRPTR détermine l'adresse du premier octet du descripteur qui en compte 6. Dans l'exemple, on détermine l'adresse de début et la longueur de A$.

FONCTION **ASC**

Syntaxe : ASC(x$)

Exemple :
```
A$="GfA"
PRINT ASC(A$)
PRINT ASC("!")
PRINT ASC("!ATTENTION!")
```

Fonction : Retourne le code ASCII du premier caractère de la chaîne x$.

Explication :

Les caractères disponibles sur l'ATARI ST sont ordonnés suivant le code ASCII. Il y a 256 caractères repérables par leur code compris entre 0 et 255 (les codes 0 à 31 représentent des caractères de contrôle). Une table de correspondance entre les caractères et leurs codes se trouve dans l'annexe C.

'*x$*' est une chaîne de caractères quelconque (string). La fonction ASC retourne le code ASCII du premier caractère d'une chaîne. Ainsi, ASC("!ATTENTION!") est équivalent à ASC("!"), et vaut 33. Si la chaîne x$ est une chaîne vide, la valeur retournée est 0.

FONCTION **ATN**

Syntaxe : ATN(x)

Exemple :
```
INPUT pente
RAD=ATN(pente/100)
DEGRE=RAD*180/PI
PRINT DEGRE
```

Fonction : Retourne l'arctangente de x.

Explication :

'x' est une expression numérique qui représente la valeur de la tangente d'un certain angle (en radian) que l'on veut calculer.

La fonction ATN retourne une valeur comprise entre -PI/2 et PI/2. L'angle obtenu a pour unité le radian. Si l'on veut convertir cet angle en degrés, il faut le multiplier par 180/PI (PI représente le périmètre d'un cercle de diamètre 1. Sa valeur est obtenu directement grâce à la fonction PI du BASIC GfA).

L'exemple ci-dessus calcule l'angle en degrés correspondant à une pente (pourcentage d'une route) donnée.

FONCTION BASEPAGE

Syntaxe : BASEPAGE

Exemple :

```
A=BASEPAGE
FOR I=A TO A+255
  N=PEEK(I)
  IF N>=32 AND N<128 THEN
    PRINT CHR$(N);
  ELSE
    PRINT ".";
  ENDIF
NEXT I
```

Fonction : Permet d'obtenir l'adresse de la page de base du BASIC GfA.

Explication :

A chaque programme qui est chargé en mémoire sous GEMDOS, correspond une page de base dont la taille est de 256 octets (comme depuis le CP/M 80). Dans les 128 premiers octets, on trouve des informations internes qui sont nécessaires au système d'exploitation pour gérer le programme en mémoire - un POKE dans cette zone peut souvent conduire au plantage de l'appareil au moment du QUIT. La deuxième moitié de la page de base contient la ligne de commande qui est utilisée dans diverses applications ou avec l'instruction EXEC. Avec DIR et FILES on utilise 44 octets de cette deuxième demi-page. Le reste peut par exemple être employé par des petits programmes machines pour l'interrogation du joystick. L'exemple ci-dessus permet de connaître le contenu ASCII de la Basepage.

INSTRUCTION I/O — BGET

Syntaxe : BGET [#]i,adr,nbr

Abrev. : BG

Exemple :
```
OPEN "I",#1,"SCR.DAT"
SEEK #1,8000
BGET #1,XBIOS(3)+24000,8000
CLOSE #1
```

Fonction : Lit des données sur un canal et les place dans une zone-mémoire.

Explication :

'i', 'adr' et 'nbr' sont des expressions entières. 'i' est le numéro de canal (voir OPEN). Cette instruction permet de lire 'nbr' octets et de les charger en mémoire à partir de l'adresse 'adr'. A l'inverse de BLOAD, on peut ici remplir plusieurs zones mémoires à partir d'un seul fichier. Dans l'exemple, on lit un fichier quelconque (la sauvegarde d'une image-écran par exemple) et on recharge une partie de son contenu dans la partie inférieure de la mémoire-écran (voir aussi BPUT, BSAVE et BLOAD).

FONCTION **BIN$**

Syntaxe : BIN$(x)

Exemples :
```
A=-1
B=&O22
PRINT BIN$(A)
PRINT BIN$(234)
PRINT BIN$(B)
```

Fonction : Convertit la valeur x en une chaîne de caractères représentant l'écriture binaire de x.

Explication :

'x' est un entier relatif (compris entre -2147483648 et +2147483647) écrit dans une représentation quelconque (c'est-à-dire la représentation décimale normale (sans préfixe), la représentation hexadécimale (préfixe & ou &H), la représentation octale (préfixe &O) ou la représentation binaire (préfixe &X)) ou une variable.

BIN$ convertit x en une chaîne de caractères qui est la représentation binaire de x (c'est-à-dire que la variable B de notre exemple est convertie en la chaîne "10010").

BIN$, tout comme HEX$ et OCT$, utilise la représentation interne des nombres sur 32 bits : le résultat obtenu n'est donc jamais précédé d'un signe. Ce n'est pas le cas avec la représentation décimale qui traite le signe des nombres entiers.

(voir aussi HEX$(x), OCT$(x) et STR$(x)).

INSTRUCTION I/O — BITBLT

Syntaxe : BITBLT s%(),d%(),p%()

Abrev. : BIT

Fonction : Instruction de copie de bloc.

Explication :
"s%( )" contient le 'source memory form descriptor' (description du bloc source),
"d%( )" contient le 'destination memory form descriptor' (description du bloc objet),
"p%( )" contient les coordonnées de deux rectangles de la même taille et le mode de copie (cf. PUT).

s%(0) : adresse du bloc (paire). Exemple : xbios(3)

s%(1) : largeur du bloc en pixels. Exemple : 640
doit être divisible par 16.

s%(2) : hauteur du bloc en pixels. Exemple : 400

s%(3) : largeur du bloc en mots. Exemple : 40

s%(4) : toujours 0.

s%(5) : nombre de plans de bits.

p%(0...3) : rectangle source (x0, y0, x1, y1).

p%(4...7) : rectangle objet (comme p%(0...3).

p%(8) : mode (comme avec PUT).

Attention : cette instruction peut vous poser des problèmes. Dans la plupart des cas, on peut la remplacer par PUT/GET.

INSTRUCTION I/O **BLOAD**
BSAVE

Syntaxe : BLOAD "filename"[,adresse]
BSAVE "filename",adresse,len

Abrev. : BL
BS

Exemples :
```
BSAVE "ecran",XBIOS(2),32000
CLS
BLOAD "ecran",XBIOS(2)
```

Fonction : Charge resp. sauvegarde une zone de la mémoire principale à partir de resp. sur la disquette.

Explication :

'filename' est un nom de fichier. L'utilisation des spécifications décrites sous DIR et de l'organisation hiérarchique des fichiers (\) est autorisée.

'adresse' est une expression numérique dont la valeur indique l'adresse de début de la zone-mémoire.

Si cette adresse manque lors du BLOAD, c'est l'adresse choisie lors du BSAVE qui est utilisée.

'len' est une expression numérique qui détermine la longueur de la zone-mémoire à sauvegarder. Dans l'exemple, on sauvegarde le contenu de la mémoire-écran (adresse de début : XBIOS(2)) sur disquette, puis on la recharge après avoir effacé l'écran.

INSTRUCTION BMOVE

Syntaxe : BMOVE src,dst,nbr

Abrev. : BM

Exemple : BMOVE XBIOS(3),XBIOS(3)+2,31998

Fonction : Transfert de blocs-mémoires.

Explication :

'scr', 'dst' et 'nbr' sont des expressions entières.

'src' (SouRCe) est l'adresse à partir de laquelle se trouvent les données à transférer.

'dst' (DeSTination) est l'adresse à partir de laquelle les 'nbr' (NomBRe) octets doivent être transférés. Ce transfert de bloc est très rapide, mais est sensiblement ralenti, si on utilise des nombres impairs.

BMOVE est une routine de transfert "intelligente" qui peut s'utiliser pour des zones-mémoires se chevauchant. L'exemple ci-dessus permet de décaler la mémoire-écran de deux octets.

INSTRUCTION GRAPHIQUE BOX

Syntaxe : BOX x0,y0,x1,y1

Abrev. : B

Exemples : BOX 100,100,400,300
BOX 500,300,700,100

Fonction : Trace un rectangle dont les deux sommets sur la diagonale ont pour coordonnées (x0,y0) et (x1,y1).

Explication :

L'origine des coordonnées est le coin supérieur gauche de l'écran. Les deux sommets ne doivent pas obligatoirement faire partie de l'écran (0-639,0-399 pour la haute résolution). Dans ce cas, il est évident que seule une partie du rectangle sera représenté (voir second exemple).

(voir aussi DEFLINE).

INSTRUCTION I/O **BPUT**

Syntaxe : BPUT [#]i,adr,nbr

Abrev. : BP

Exemple :
```
OPEN "O",#1,"SCR.DAT"
BPUT #1,XBIOS(3)+16000,16000
BPUT #1,XBIOS(3),16000
CLOSE #1
```

Fonction : Envoie une zone mémoire sur un canal de données.

Explication :

'i','adr' et 'nbr' sont des expressions entières.

'i' est le numéro d'un canal de données (voir OPEN). L'instruction BPUT lit les 'nbr' octets mémorisés à partir de l'adresse 'adr', et les envoie sur le canal 'i'. A l'inverse de BSAVE, on peut ici écrire plusieurs zones mémoires sur le même fichier. Dans l'exemple ci-dessus, on sauvegarde la mémoire-écran sur fichier, en intervertissant les deux demi-écrans.

(voir aussi BGET, BSAVE, BLOAD).

FONCTION **C:**

Syntaxe : C:var(listeparam)

Exemple :
```
Q%=VARPTR(Q$)
A=C:Q%(17,L:0,W:-1)
```

Fonction : Appel d'un sous-programme en langage machine (langage C compilé) avec passage de paramètres comme en C.

Explication :

'var' est une variable numérique (mais pas une variable booléenne) qui contient l'adresse de début de la routine C.

'listeparam' est - comme en C - une liste de paramètres qui sont soit des mots (16 bits=W) soit des mots longs (32 bits=L). La liste peut aussi être vide. Pour transmettre des mots longs, il faut utiliser le préfixe 'L:'. Pour les mots, on peut employer le préfixe 'W:' qui est facultatif. Au retour de la routine C, le BASIC GfA (comme le C) collecte le mot long contenu dans le registre D0. L'exemple sert uniquement à mettre en évidence le format de l'instruction.

Dans l'exemple, on appelle une routine qui serait contenue dans la chaîne de caractères Q$.

(voir aussi BIOS, XBIOS et GEMDOS).

INSTRUCTION CALL

Syntaxe : CALL var
CALL var(listeparam)

Abrev. : CA

Exemple : CALL Test(3,i%,a$)

Fonction : Appel d'un programme-machine.

Explication :

'*var*' est une variable numérique (mais non booléenne) qui contient l'adresse de début du programme-machine.

'*listeparam*' est une liste de paramètres qui sont transmis à la routine machine. Le programme-machine obtiendra sur la pile le nombre de paramètres (contenu dans un mot de 16 bits) et l'adresse d'un tableau (contenu dans un mot long de 32 bits) qui contient chaque paramètre sur quatre octets. Les nombres sont transmis dans des mots longs ; en ce qui concerne les chaînes de caractères, on transmet leur adresse de début. L'exemple est uniquement là pour mettre en évidence le format de l'instruction.

INSTRUCTION I/O — CHAIN

Syntaxe : CHAIN "nomfich"

Abrev. : CH

Exemple :
```
SAVE "PROGRAM"
CHAIN "PRO*"
```

Fonction : Charge un programme en mémoire principale et lance son exécution.

Explication :

'*nomfich*' est le nom du fichier à chainer.

Pour ce nom de fichier, toutes les spécifications sont autorisées (voir DIR). Si on ne donne pas d'extension au nom du programme, le suffixe .BAS est rajouté par défaut par le BASIC GfA. Les guillemets autour de 'nomfich' sont automatiquement mis en place, s'ils n'ont pas été entrés au clavier.

INSTRUCTION I/O **CHDIR**

Syntaxe : CHDIR "nomrepert"

Abrev. : CHD

Exemple :
```
MKDIR "\REP1"
MKDIR "\REP1\REP2"
CHDIR "\"
FILES
CHDIR "\REP1"
FILES
```

Fonction : Permet de changer de répertoire (directory).

Explication :

'*nomrepert*' est le nom du répertoire sous lequel on veut se placer.

La syntaxe de ce nom dépend de la place hiérarchique occupée par le répertoire : CHDIR "\" ramène au répertoire principal (racine).

Lors d'un changement vers n'importe quel autre répertoire, il est nécessaire d'introduire le chemin. Celui-ci commence toujours par '\' quand on part du répertoire principal. Ainsi, par exemple, CHDIR "\REP1\REP2" permet de se placer sous le répertoire REP2 lui-même contenu dans le répertoire REP1 de la directory principale. Un changement de lecteur n'est pas possible avec cette instruction, mais est réalisable avec l'instruction CHDRIVE.

INSTRUCTION **CHDRIVE**

Syntaxe : CHDRIVE n

Abrev. : CHDR

Exemple :
```
CHDRIVE 1
FILES
CHDRIVE 2
FILES
```

Fonction : Détermine l'unité de disquettes prise par défaut.

Explication :

'n' est une expression numérique qui indique le numéro (1-15) de l'unité de disquettes.

CHDRIVE détermine quel est l'unité de disquettes à utiliser lorsqu'aucune spécification concernant le lecteur (par ex. : A:) n'est précisée dans une instruction ou une fonction qui effectue des accès-disquettes.

L'exemple affiche l'une après l'autre les directories des disquettes de l'unité A et B.

ATTENTION : **CHDRIVE 0 entraine le plantage du système.**

FONCTION **CHR$**

Syntaxe : CHR$(x)

Exemples :
```
PRINT CHR$(65)
PRINT CHR$(577)
PRINT CHR$(-191)
```

Fonction : Retourne le caractère dont le code ASCII est x.

Explication :

Les caractères disponibles sur l'ATARI ST sont numérotés suivant le code ASCII. Il y a 256 caractères dont les codes sont compris entre 0 et 255 (les codes 0 à 31 représentent des caractères de contrôle). Une table de correspondance entre les différents caractères et leurs codes se trouve en annexe C.

'*x*' est un entier relatif. 'x' est tout d'abord converti en un entier compris entre 0 et 255 grâce à la formule x AND 255.

On obtient une chaîne de caractères de longueur 1 avec le caractère correspondant au code ASCII. Ainsi les trois exemples ci-dessus renvoient tous la chaîne de caractères "A".

INSTRUCTION GRAPHIQUE CIRCLE

Syntaxe : CIRCLE x,y,r[,phi0,phi1]

Abrev. : C

Exemples : CIRCLE 320,200,200,900,2700
CIRCLE 700,500,400

Fonction : Trace un [arc de] cercle de centre (x,y) et de rayon r.

Explication :

x et y sont les coordonnées du centre du cercle, l'origine des axes étant le coin supérieur gauche de l'écran. Bien que l'écran ne comporte que 640 x 400 points (dans le cas de la haute résolution), il est possible de donner à ces coordonnées des valeurs plus grandes ou plus petites. Le centre du cercle se trouve alors naturellement en dehors de l'écran.

Si on veut uniquement représenter un arc de cercle, on introduit en plus les valeurs de l'angle initial (phi0) et de l'angle final (phi1). Cette angle est donné en 1/10 de degré. Un angle de 0 degré (0) est dirigé vers la droite, un angle de 90 degrés (900) vers le haut, un angle de 180 degrés (1800) vers la gauche et un angle de 270 degrés (2700) vers le bas.

(voir aussi DEFLINE).

INSTRUCTION I/O — CLEAR

Syntaxe : CLEAR

Abrev. : CLE

Exemple :

```
A=17
B$="BASIC"
CLEAR
PRINT A,B$
```

Fonction : Efface toutes les variables et tous les tableaux.

Explication :

Toutes les variables numériques sont remises à 0 et les variables chaînes de caractères deviennent des chaînes vides. Le programme en lui-même n'est pas modifié.

L'instruction CLEAR ne peut être employée à l'intérieur de procédures ou de boucles FOR-NEXT.

INSTRUCTION CLEARW

Syntaxe : CLEARW n

Abrev. : CLE W

Exemple : CLEARW 2

Fonction : Efface le contenu de la fenêtre numéro n.

Explication :

'*n*' est une expression numérique dont la valeur représente le numéro de la fenêtre.

INSTRUCTION I/O **CLOSE**

Syntaxe : CLOSE [[#]n]

Abrev. : CL

Exemple :

```
OPEN "O",#1,"PRO"
PRINT #1,"GfA-BASIC"
CLOSE #1
OPEN "I",#1,"PRO"
INPUT #1,A$
CLOSE
PRINT A$
```

Fonction : Ferme un canal de données ou un canal lié à un fichier sur disquette.

Explication :

'n' est un entier compris entre 0 et 99 et indique le numéro du fichier qui a été ouvert par OPEN et qui doit maintenant être fermé. Si l'argument facultatif 'n' est absent, tous les canaux qui étaient encore ouverts sont refermés.

Avec l'instruction OPEN, on attribue un numéro à un fichier. Ce numéro permet d'atteindre le fichier pour les instructions de lecture et/ou d'écriture. La correspondance entre le fichier et son numéro cesse avec l'instruction CLOSE.

INSTRUCTION CLOSEW

Syntaxe : CLOSEW n

Abrev. : CL W

Exemple : CLOSEW 2

Fonction : Ferme la fenêtre numéro n.

Explication :

'n' est une expression numérique dont la valeur représente le numéro d'une fenêtre. CLOSEW 0 permet de revenir à l'affichage normal sur l'écran. Cette instruction est indispensable après la fin de l'utilisation d'une fenêtre.

INSTRUCTION **CLR**

Syntaxe : CLR var[,var...]

Exemple : CLR A,B,C%,D!,ES

Fonction : Efface une liste de variables.

Explication :

'var' représente un nom de variable (pas de tableau). L'exemple produit le même effet que :

```
A=0
B=0
C%=0
ES=""
```

INSTRUCTION **CLS**

Syntaxe : CLS [#i]

Abrev. : CLS

Exemple :

```
FOR I=1 TO 1999
  PRINT "A";
NEXT I
PRINT AT(32,13);" Appuyez sur une touche ..."
A=INP(2)
CLS
```

Fonction : Efface l'écran.

Explication :

L'instruction CLS est identique à PRINT CHR$(27);"E"; et entraine l'effacement de l'écran. Le curseur est replacé à sa position HOME (en haut et à gauche de l'écran). CLS peut aussi être utilisé pour des canaux de données (qui ont par exemple été ouvert par OPEN "",#1,"VID:"), la syntaxe est alors CLS #i (où i indique le numéro du canal). Dans ce cas, l'instruction est identique à PRINT #1,CHR$(27);"E"; .

INSTRUCTION GRAPHIQUE COLOR

Syntaxe : COLOR c

Abrev. : CO

Exemple :
```
PCIRCLE 100,100,100
COLOR 0
LINE 0,0,200,200
```

Fonction : Met en place la couleur.

Explication :

Cette instruction permet de choisir la couleur pour les instructions BOX, CIRCLE, DRAW, ELLIPSE, LINE, PLOT et RBOX.

'c' indique le numéro de la couleur voulue (en basse résolution, la valeur de la couleur varie entre 0 et 15, en moyenne résolution, entre 0 et 3 et en haute résolution, entre 0 et 1).

INSTRUCTION CONT

Syntaxe : CONT

Abrev. : CON

Fonction : Relance l'exécution à partir du point d'arrêt.

Explication :

Cette instruction n'a de sens qu'en mode direct. Elle permet la reprise de l'exécution du programme après une interruption de celle-ci par l'instruction STOP ou l'emploi simultané des touches <CONTROL>, <SHIFT> et <ALTERNATE>. CONT n'est plus possible après une modification du programme, après l'instruction CLEAR ou l'introduction de nouvelles variables.

FONCTION COS

Syntaxe : COS(x)

Exemples :

1)
```
INPUT RAD
PRINT COS(RAD)
INPUT DEG
PRINT COS(DEG*PI/180)
```

2)
```
PLOT 320,200
FOR I=1 TO 5400
  X=I/36*COS(I*PI/180)
  Y=I/36*SIN(I*PI/180)
  DRAW TO 320+X,200+Y
NEXT I
```

Fonction : Retourne le cosinus de x.

Explication :

'x' est une expression numérique qui représente l'angle (en radians) dont on veut calculer le cosinus.

Si l'on veut introduire l'angle en degré, il faut remplacer x par deg*PI/180, où PI est une fonction du BASIC GfA qui renvoie la valeur de pi (c'est-à-dire le périmètre d'un cercle de diamètre 1).

Le premier exemple calcule tout d'abord le cosinus d'un angle donné en radians, puis celui d'un angle donné en degrés. Le deuxième exemple dessine une spirale degré par degré.

	FONCTION	CRSCOL CRSLIN

Syntaxe :

```
CRSCOL
CRSLIN
```

Exemple :

```
Ox=CRSCOL
Oy=CRSLIN
PRINT AT(9,9);"TEST"
PRINT AT(Ox,Oy);
```

Fonction : Permet d'obtenir la position du curseur (colonne et ligne).

Explication :

Les fonctions CRSLIN resp. CRSCOL retournent la ligne resp. la colonne où se trouve le curseur. Dans l'exemple, on sauve la position du curseur, on affiche ensuite un message et enfin, on replace le curseur à sa position initiale.

(voir aussi PRINT AT).

FONCTIONS **CVI**
CVL
CVS
CVF
CVD

Syntaxe :
CVI(x$)
CVL(x$)
CVS(x$)
CVF(x$)
CVD(x$)

Exemple :
```
A=0.1111
A$=MKF$(A)
PRINT CVF(A$)
```

Fonction : Convertit une chaîne de caractère en une valeur numérique.

Explication :

'x$' est une chaîne de caractères quelconque.
Les différentes fonctions convertissent x$ en une valeur numérique, à savoir :

* CVI convertit une chaîne de 2 octets en un entier de 16 bits.
* CVL convertit une chaîne de 4 octets en un entier de 32 bits.
* CVS convertit une chaîne de 4 octets réalisant le format compatible-BASIC Atari en une valeur numérique.
* CVF convertit une chaîne de 6 octets réalisant le format spécifique au BASIC GfA en une valeur numérique.
* CVD convertit une chaîne de 8 octets réalisant le format compatible-MBASIC en une valeur numérique. Si la chaîne de caractères est plus longue que le format permis, seul est pris en compte la sous-chaîne de longueur correspondante. Si la chaîne est plus courte, la valeur numérique obtenu est 0.
  (fcts inverses : MKI$, MKL$, MKS$, MKF$, MKD$)

INSTRUCTION DATA

Syntaxe : DATA [const[,const]...]

Abrev. : D

Exemple :
```
READ A,B$,CS,D,E!
PRINT A;B$;C$;D;E!
DATA 234,"G,f,A",  BASIC,&HFF,56
```

Fonction : Permet de stocker des valeurs qui pourront être lues par READ.

Explication :

'const' est une constante numérique, booléenne ou alphanumérique. Les constantes sont séparées entre elles par des virgules. Les constantes numériques peuvent être représentées sous n'importe quel format (décimal, héxadécimal, octal ou binaire). Les constantes alphanumériques (chaînes de caractères) n'ont pas besoin d'être placées entre guillemets ; ceux-ci sont cependant nécessaires si la chaîne comporte une ou plusieurs virgules. Si on n'utilise pas les guillemets, la chaîne de caractères va de virgule en virgule, cela signifie que les blancs devant ou derrière une virgule sont pris en compte dans la chaine de caractères (cf exemple). Si l'instruction DATA n'est suivie d'aucune valeur, cela est interprété comme une chaîne de caractères vide. Si on lit, avec l'instruction READ, la valeur 0 pour une variable booléenne, on affecte à celle-ci 0. Pour toutes les autres valeurs, on affecte à la variable booléenne -1.

(voir aussi READ et RESTORE).

FONCTION **DATE$**

Syntaxe : DATE$

Exemple :
```
DEFTEXT 1,16,0,32
PRINT DATE$
TEXT 240,180,DATE$
```

Fonction : Retourne la date.

Explication :

La fonction retourne une chaîne de caractère donnant la date mémorisée dans le tableau de contrôle, sous le format suivant :

- **jj.mm.aaaa**

INSTRUCTION ARITHMETIQUE DEC

Syntaxe : DEC var

Abrev. : DEC

Exemple :

```
T=TIMER
FOR I%=1 TO 10000
  DEC A%
NEXT I%
PRINT (TIMER-T)/200
A%=0
T=TIMER
FOR I%=1 TO 10000
  A%=A%-1
NEXT I%
PRINT (TIMER=T)/200
```

Fonction : Diminue var de 1.

Explication :

'*var*' doit être une variable numérique ou un élément de tableau numérique. DEC var est identique à var=var-1. L'avantage de l'instruction DEC réside dans son format concis (économie de place mémoire) et surtout dans sa vitesse d'exécution (testez l'exemple !). Ainsi, l'utilisation de l'instruction DEC permet d'augmenter de façon sensible la vitesse d'exécution qui par ailleurs est déjà très élevée (dans l'exemple, la vitesse est environ multipliée par 3,5).

INSTRUCTION **DEFNUM**

Syntaxe : DEFNUM n

Abrev. : DEFN

Exemple :
```
DEFNUM 8
FOR I=-1 TO 1 STEP 0.1
  PRINT I
NEXT I
```

Fonction : Arrondit tous les nombres à n chiffres, avant leur affichage.

Explication :

'*n*' est un nombre compris entre 3 et 11 qui indique la précision à laquelle les nombres vont être affichés. Cette instruction est utile pour afficher les valeurs exactes de nombres qui sont en réalité légèrement imprécis à cause des erreurs d'arrondi. Dans l'exemple, on affiche les multiples exacts de 0.1 à l'écran, alors que les opérations avec la valeur 0.1 conduisent en réalité bien souvent à une imprécision.

Par ailleurs, il est conseillé dans les programmes financiers de calculer en centimes, car les nombres entiers n'entraînent pratiquement jamais des erreurs d'arrondi. De plus, l'utilisation des variables Integer (Var%) permet d'augmenter sensiblement la vitesse d'exécution de l'interpréteur.

(Voir aussi PRINT USING).

INSTRUCTION GRAPHIQUE DEFFILL

Syntaxe : DEFFILL [c],[a],[b] ou
DEFFILL [c],A$

Abrev. : DEFF

Exemple :
```
BOX 100,100,400,300
DEFFILL 1,2,9
FILL 50,200
DEFFILL ,,10
FILL 320,200
```

Fonction : Définit la couleur et le motif de remplissage, ou permet de redéfinir soi-même un motif.

Explication :

Les instructions *FILL, PBOX, PCIRCLE, PELLIPSE et PRBOX* permettent respectivement de remplir des formes ou de dessiner des figures pleines avec un motif de remplissage.

L'instruction *DEFFILL* permet de définir la couleur et le motif.

'*c*' détermine la couleur et '*a*','*b*' le motif.
Les virgules en fin d'instruction, dans le cas où a et (ou) b sont absents, peuvent disparaître.

'*a*' indique le mode de remplissage (0=vide, 1=plein, 2=pointillé, 3=hachuré, 4=redéfini soi-même).

'*b*' permet de choisir entre 24 motifs différents parmi les pointillés et entre 12 motifs parmi les hachures.
Vous trouverez une table des motifs deux pages plus loin. ==>

INSTRUCTION GRAPHIQUE DEFFILL
suite

La deuxième variante de l'instruction DEFFILL permet de redéfinir soi-même le motif de remplissage. Le motif est défini par une matrice de 16x16 points. La chaîne de définition est la concaténation (avec +) de 16 séries de bits transformées en chaînes de caractères par MKI$, une série de bits correspondant à une ligne. Ainsi, la chaîne de caractères définie par :

```
FOR I=1 TO 16
  A$=A$+MKI$(65535)
NEXT I
```

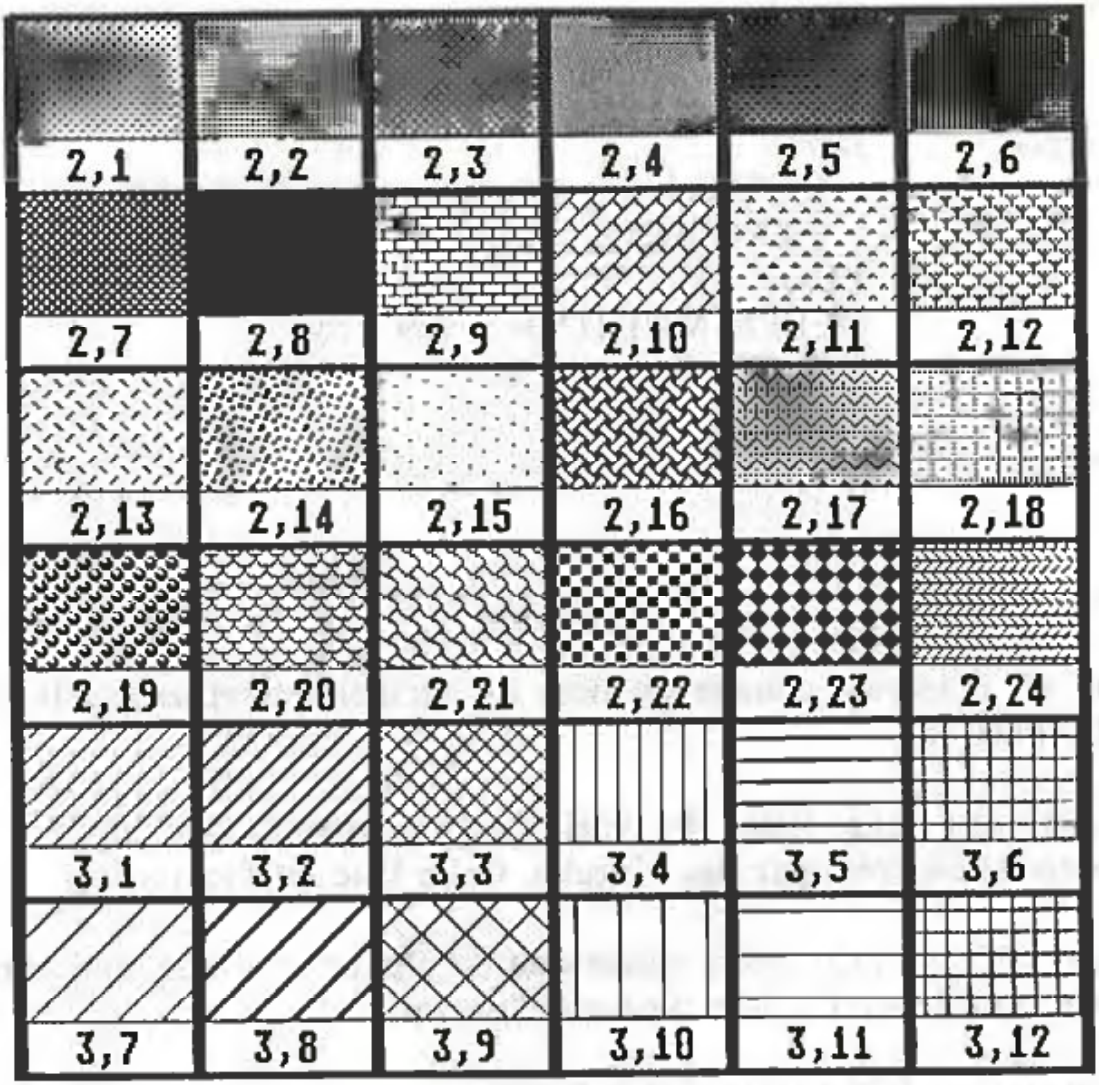
correspondrait au motif de remplissage plein, car 65535 donne en binaire :

1111111111111111 .

==>

INSTRUCTION GRAPHIQUE DEFFILL
suite

TABLE DES MOTIFS



INSTRUCTION **DEFFN**

Syntaxe : DEFFN nom[(listevar)]=expres
Appel par :
FN nom[(listeexpres)]

Abrev. : DEFFN
(@ peut remplacer FN)

Exemple :
```
DO
  INPUT X
  PRINT X,FN Trois, FN MULT(10)
LOOP
DEFFN MULT(Y)=Y*FN Trois
DEFFN Trois = 3*X
```

Fonction : Définit une fonction spécifique de l'utilisateur.

Explication :

'nom' est construit comme un nom de variable et représente le nom de la fonction.

'listevar' est une liste de variables (également des chaînes de caractères) séparées par des virgules. Cette liste est facultative.

'expres' est une expression numérique ou alphanumérique quelconque. ('nom' doit être du même type que l'expression).

L'instruction *DEFFN* permet à l'utilisateur de définir soi-même une fonction. Cette fonction peut être appelée par FN suivi du nom de la fonction correspondant et éventuellement par une liste de paramètres 'listeexpres' de même type que les variables correspondantes de 'listevar'. Lors de l'appel de la fonction par FN, la présence de 'listeexpres' est obligatoire dès lors que 'listevar' existe. ==>

INSTRUCTION **DEFFN suite**

Lors de l'appel de la fonction par FN ou par @, toutes les variables de 'expres' prennent les valeurs des expressions de 'listeexpres' afin de calculer et de retourner la valeur de la fonction.

Il est possible, à l'intérieur d'une définition de fonction, de faire appel à une autre fonction (voir exemple). On peut ainsi définir des fonctions dont la taille dépasse la longueur d'une ligne.

Il n'est pas nécessaire de faire figurer les instructions DEFFN en début de programme, car l'interpréteur exécute tous les DEFFN au début de l'exécution du programme. De même, l'ordre des instructions DEFFN n'a pas d'importance, même si, dans une définition de fonction, on appelle une fonction définie plus loin (voir exemple).

INSTRUCTION GRAPHIQUE **DEFLINE**

Syntaxe : DEFLINE [t],[l],[d],[f]
(les virgules finales peuvent disparaître).

Abrev. : DE

Exemple :

```
DEFLINE 1,10,2,1
LINE 140,25,350,25
PRINT AT(1,2);"a=2, e=1, b=10:"
FOR I=1 TO 6
  PRINT "s=";I;":"
  DEFLINE I,1,0,0
  LINE 50,25+I*16,350,25+I*16
NEXT I
```

Fonction : Définit les caractéristiques des lignes.

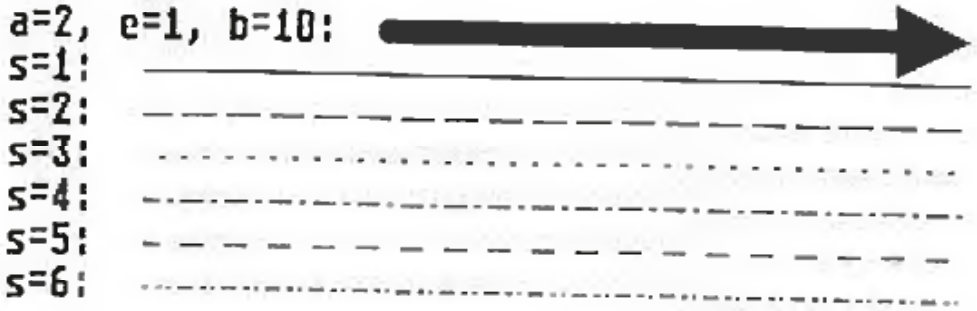
Explication :

Cette instruction permet de définir les types de lignes qui seront utilisées par BOX, CIRCLE, DRAW, ELLIPSE, LINE et RBOX.

'l' indique la largeur du trait en nombre de points graphiques.

'd' et 'f' indiquent le type d'extrémité à utiliser respectivement au début et à la fin de la ligne (0= normal, 1= flèche, 2=arrondi).

Enfin, *'t'* détermine le type de trait. Le programme de notre exemple donne le résultat suivant :



INSTRUCTION **DEFLIST**

Syntaxe : DEFLIST x

Abrev. : DEFLIS

Exemple : DEFLIST 0

Fonction : Définit le format des listings.

Explication :

'x' est une expression numérique quelconque.

Cette instruction permet de définir le format des listings de programmes.

Pour x=0 :
Les noms des instructions et des fonctions sont entièrement écrits en majuscules, les variables par contre sont représentées en minuscules.

Ex : PRINT a,bal,CHR$(65)

Pour x<>0 :
Les noms des instructions, des fonctions et des variables sont écrits en minuscules sauf la première lettre qui est une majuscule.

Ex : Print A,Bal,ChrS(65)

INSTRUCTION GRAPHIQUE **DEFMARK**

Syntaxe : DEFMARK [c],[s],[t]

Abrev. : DEFM

Exemple :

```
DIM X(5),Y(5)
DO
  FOR I=0 TO 5
    X(I)=RANDOM(640)
    Y(I)=RANDOM(400)
  NEXT I
  DEFMARK 1,RANDOM(7),0
  POLYMARK 6,X(),Y()
LOOP
```

Fonction : Définit la couleur, le type et la taille des sommets qui seront placés par l'instruction POLYMARK.

Explication :

'*c*' est le numéro de la couleur choisie (0 ou 1 en haute résolution).
'*s*' détermine le symbole qui sera employée. On a le choix entre les 6 symboles :

1 = Point
2 = Plus
3 = Etoile
4 = Rectangle
5 = Croix
6 = Losange

Toute autre valeur de 's' donne une étoile.
'*t*' définit la taille du symbole. Les valeurs 0,20,40,60,80,100 sont possibles.

INSTRUCTION GRAPHIQUE DEFMOUSE

Syntaxe : DEFMOUSE n ou
DEFMOUSE A$

Abrev. : DEFMO

Exemple :

```
FOR I=0 TO 7
  DEFMOUSE I
  PAUSE 100
NEXT I
X$=MKI$(1)
Y$=MKI$(1)
MF$=MKI$(0)
CF$=MKI$(1)
M$=MKI$(65535)
FOR I=1 TO 14
  M$=M$+MKI$(32769)
NEXT I
M$=M$+MKI$(65535)
C$=MKI$(0)
FOR I=1 TO 14
  C$=C$+MKI$(36766)
NEXT I
C$=C$+MKI$(0)
A$=X$+Y$+MKI$(1)+MF$+CF$+M$+C$
PBOX 10,10,200,200
DEFMOUSE A$
DO
LOOP
```

Fonction : Permet de choisir ou de redéfinir une forme de souris.

Explication :

L'instruction DEFMOUSE n permet de choisir l'une des 8 formes de souris prédéfinies. **'n'** est une expression numérique dont la valeur est comprise entre 0 et 7 (pour des valeurs supérieures, n mod 8 est pris en compte). ==>

INSTRUCTION GRAPHIQUE DEFMOUSE
suite

Les formes suivantes sont disponibles :

n=0	Flèche
n=1	X allongé (arrondi)
n=2	Abeille
n=3	Main avec doigt pointé
n=4	Main plate
n=5	Réticule fin
n=6	Réticule épais
n=7	Réticule avec contour

L'instruction DEFMOUSE A$ permet de redéfinir soi-même une forme de souris. Pour le dessin de la souris, on dispose de 16x16 points. Sur cette matrice, il est nécessaire de définir deux figures : le masque et le curseur de la souris lui-même. Cela est nécessaire pour que le curseur reste visible, lorsqu'il est déplacé sur une surface de même couleur. L'un des 256 points représente le point d'action, qui est pris en compte par les différentes fonctions de la souris (comme MOUSE par exemple).

La chaîne de définition A$ est composée de la manière suivante :

```
A$ = MKI$(coord. x du point d'action)
   + MKI$(coord. y du point d'action)
   + MKI$(1)
   + MKI$(couleur du masque)  hab. 0
   + MKI$(couleur du curseur)  hab. 1
   + M$      (motif binaire du masque)
   + C$      (motif binaire du curseur)
```

Les deux motifs binaires sont composés de 16 mots (MKI$) qui indique chacun le motif binaire d'une ligne. Ainsi, MKI$(65535) représente une ligne pleine, car 65535 donne en binaire :

1111111111111111 .

Pour mieux comprendre cette instruction, il est conseillé de mettre en pratique l'exemple.

INSTRUCTION GRAPHIQUE DEFTEXT

Syntaxe : DEFTEXT [c],[t],[a],[h]
(les virgules finales peuvent disparaître).

Abrev. : DEFT

Exemple :
```
DEFTEXT 1,16,0,32
TEXT 10,50,"contour"
DEFTEXT ,24,900,
TEXT 500,300,"contour/souligne"
```

Fonction : Définit la couleur, le type, l'orientation et la taille des caractères affichés par l'instruction TEXT.

Explication :

'c' est le numéro de la couleur choisie (0 ou 1 en haute résolution).

't' indique le type de caractère. Voici la signification de certaines des valeurs que peut prendre t : 0=normal, 1=gras, 2=clair, 4=oblique, 8=souligné, 16=contour. Si l'on souhaite avoir une écriture contour et soulignée, on additionne les types fontamentaux ci-dessus pour obtenir 8+16 -> t=24.

'a' donne l'angle d'orientation de l'écriture en 1/10 de degré. Il n'y a que 4 valeurs possibles : 0 degré (0), 90 degrés (900), 180 degrés (1800) et 270 degrés (2700).

'h' indique la hauteur des caractères en nombre de points : 4=écriture petite (Icônes), 6=écriture en couleur (8*8), 13=écriture normale (noir et blanc), 32=écriture énorme.

FONCTION DFREE

Syntaxe : DFREE(n)

Exemples :
```
PRINT DFREE(0)
PRINT DFREE(1)
```

Fonction : Indique la place mémoire disponible sur une disquette.

Explication :

'*n*' est une expression numérique qui correspond au numéro de l'unité de disquettes.

L'instruction retourne la place mémoire disponible sur la disquette de l'unité n.

Si n=0, il s'agit de l'unité de disquette utilisée actuellement.

INSTRUCTION **DIM**

Syntaxe : DIM var(indices)[,var(indices),...]

Abrev. : DI

Exemples : DIM A(1000),B(4,5,3)
DIM N$(20,5)

Fonction : Détermine les dimensions d'un tableau.

Explication :

'var' est une variable numérique, alphanumérique ou booléenne quelconque.

'indices' est une liste d'expressions numériques séparées par des virgules et indiquant les dimensions du tableau.

On peut par exemple se représenter un tableau de dimension 3 comme un immeuble de bureaux, pour lequel le premier indice serait l'étage, le deuxième indice le numéro de chambre d'un étage déterminé et le troisième indice le numéro de table d'une chambre déterminé.

Transposé en BASIC, cela signifie qu'on peut accéder à différentes places de la mémoire (des tables dans notre exemple) grâce à la connaissance d'un certain nombre d'indices.

L'instruction DIM détermine la borne supérieure de l'indice (la valeur la plus petite que peut prendre l'indice est 0) et réserve une zone mémoire de taille correspondante.

Pour les tableaux à plusieurs dimensions, le nombre maximal d'éléments est de 65535. Quant aux tableaux à une dimension, ils ne sont limités que par la taille de la mémoire principale.

FONCTION **DIM?**

Syntaxe : DIM?(tableau())

Exemples :
```
DIM A$(3,4,5)
DIM N%(12,12)
PRINT DIM?(A$())
PRINT DIM?(N%())
```

Fonction : Retourne le nombre d'éléments d'un tableau.

Explication :

'tableau' est une variable tableau quelconque.

La fonction donne le nombre d'éléments d'un tableau qui a précédemment été défini avec l'instruction DIM. Notez que les indices commencent à 0 : ainsi, l'exemple ci-dessus affiche les valeurs 120 et 169.

Si on demande la dimension d'un tableau non défini, on obtient en retour la valeur 0.

INSTRUCTION I/O DIR

Syntaxe : DIR["specfich"[TO "fich"]]

Abrev. : DIR

Exemples : DIR "A:*.*" TO "A:TABMAT"
DIR "A:*.*" TO "LST:"

Fonction : Liste les fichiers d'une disquette.

Explication :

'*specfich*' est le nom d'un fichier. Toutes les spécifications suivantes sont permises :

L'unité de disquettes est spécifiée par une lettre suivie d'un double point, placée devant le nom du fichier (par ex. A: pour l'unité 1).

Si le nom du fichier de 'specfich' comporte un (ou plusieurs) points d'interrogation, on prend en compte tous les fichiers dont les noms coïncident, le point d'interrogation remplaçant une lettre quelconque. L'instruction DIR 'ART??.DOC', par exemple, recherche tous les fichiers ayant l'extension .DOC et un nom de 5 lettres commençant par les 3 lettres ART . Les fichiers ARTNR.DOC et ARTEN.DOC sont donc tous deux pris en compte.

Le caractère '*' signifie que toute une partie du nom peut être quelconque. Ainsi, DIR '*.DOC' recherche tous les fichiers avec l'extension .DOC, et DIR 'B*.*' les fichiers commençant par la lettre B.

INSTRUCTION I/O **DIR**
suite

L'instruction DIR permet aussi d'utiliser le système de hiérarchie des fichiers (\) d'UNIX (ou MS-DOS). Ainsi, l'instruction DIR '\KFZ\TNR*.*' recherchera tous les fichiers du répertoire 'TNR', lui-même contenu dans le répertoire 'KFZ', lui-même contenu dans le répertoire principal.

Grâce au terme supplémentaire facultatif TO "fich", il est possible d'imprimer un listing des fichiers recherchés ("LST:") ou de sauvegarder cette même liste sur un fichier quelconque.

Ainsi, l'exemple ci-dessus va créer, sur la disquette de l'unité A, un fichier de nom 'TABMAT' qui contiendra tous les fichiers de cette disquette, et sort la nouvelle liste des fichiers (qui comprend maintenant le fichier de nom "TABMAT") sur imprimante.

FONCTION **DIR$**

Syntaxe : DIR$(n)

Exemple :
```
MKDIR "\TEST"
CHDIR "\TEST"
PRINT DIR$(0)
```

Fonction : Donne le nom du répertoire actif de l'unité de disquettes numéro **n**.

Explication :

'*n*' est une expression numérique dont la valeur indique le numéro du lecteur de disquettes (1=A:, 2=B:,...0=lecteur utilisé).

Cette fonction retourne le nom du répertoire actif pour l'unité considérée. S'il s'agit de la directory principale, la fonction renvoie une chaîne de caractères vide.

INSTRUCTION ARITHMETIQUE DIV

Syntaxe : DIV var,n

Abrev. : DIV

Exemple :

```
DIM A%(10000)
FOR I=1 TO 10000
  A%(I)=5
NEXT I
T=TIMER
FOR I=1 TO 10000
  DIV A%(I),5
NEXT I
PRINT (TIMER-T)/200
T=TIMER
FOR i=1 TO 10000
  A%(I)=A%(I)/5
NEXT I
PRINT (TIMER-T)/200
```

Fonction : Divise la valeur de var par n

Explication :

'*var*' doit être une variable numérique ou un élément de tableau numérique. '**n**' est une expression numérique.

DIV var,n est identique à var=var/n. L'avantage de l'instruction DIV réside dans la vitesse d'exécution (il suffit de tester l'exemple !). De cette manière, la vitesse d'exécution déjà très élevée peut encore être augmentée (d'environ 30% dans l'exemple).

INSTRUCTION DE STRUCTURE DO...LOOP

Syntaxe :
```
DO
LOOP
```

Abrev. :
```
DO
L
```

Exemple :
```
DO
  INC I
  PRINT I
LOOP
```

Fonction : Met en place une boucle sans fin.

Explication :

La partie du programme qui se trouve entre DO et LOOP est répétée à l'infini.

On peut sortir de cette boucle par l'instruction EXIT. Si cette dernière instruction n'est pas utilisée, il ne reste plus qu'une solution pour interrompre le programme : l'action simultanée sur les touches <CONTROL>, <SHIFT> et <ALTERNATE>.

INSTRUCTION GRAPHIQUE DRAW

Syntaxe : DRAW [TO] x0,y0
DRAW x0,y0 TO x1,y1 [TO x2,y2...]

Abrev. : DR

Exemples :
```
DRAW 10,10
DRAW TO 200,10
DRAW 200,10 TO 200,300
DRAW 200,300 TO 10,300 TO 10,10
```

Fonction : Affiche un point ou relie deux ou plusieurs points entre eux par des droites.

Explications :

(x0,y0), (x1,y1),... sont les coordonnées de points, l'origine des axes étant le coin supérieur gauche de l'écran.

L'instruction DRAW x,y affiche un point et est identique à l'instruction PLOT x,y.

DRAW TO x,y relie le point (x,y) avec le dernier point qui a été affiché (par PLOT, LINE ou DRAW).

DRAW x0,y0 TO x1,y1 relie les points (x0,y0) et (x1,y1) par une droite, et est identique à l'instruction LINE x0,y0,x1,y1.

DRAW x0,y0 TO x1,y1 TO x2,y2 TO... relie dans l'ordre les points (x0,y0), (x1,y1), (x2,y2),... par des droites.

L'exemple permet de tracer un rectangle.

INSTRUCTION **EDIT**

Syntaxe : EDIT

Abrev. : ED

Exemple :
```
PRINT "Retour à l'éditeur sans"
PRINT "boîte d'alarme"
EDIT
```

Fonction : Retour à l'éditeur.

Explication :

Utilisé en mode direct, cette instruction permet de revenir à l'éditeur. Dans un programme, l'instruction EDIT a presque le même effet que END, avec la différence que la main est rendue à l'éditeur sans passer par la boîte de fin de programme.

INSTRUCTION GRAPHIQUE ELLIPSE

Syntaxe : ELLIPSE x,y,rx,ry[,phi0,phi1]

Abrev. : ELL

Exemples : ELLIPSE 320,200,300,250
ELLIPSE 320,0,320,400,1800,3600

Fonction : Trace une ellipse (ou un arc d'ellipse) de centre (x,y) et de demi-axes rx et ry.

Explication :

Le coin supérieur gauche sert d'origine pour le centre de coordonnées (x,y).

Le centre et les demi-axes peuvent être choisis de telle sorte que l'ellipse n'apparaisse pas entièrement sur l'écran.

'rx' est la longueur du demi-axe horizontal et ry celle du demi-axe vertical.

Si l'on veut uniquement tracer un arc d'ellipse, on définit l'angle de départ phi0 et l'angle final phi1. Ces angles doivent être donnés en 1/10 de degrés, sachant que 0 degré (0) se trouve à droite, 90 degrés (900) en haut, 180 degrés (1800) à gauche, 270 degrés (2700) en bas et 360 degrés (3600) à nouveau à droite.

(voir aussi DEFLINE).

INSTRUCTION	END

Syntaxe : END

Abrev. : END

Exemple :

```
DO
  INC I
  IF  I=10000 THEN
     END
  ENDIF
LOOP
```

Fonction : Ferme tous les fichiers et termine l'exécution du programme.

Explication :

Les instructions END peuvent être placées partout dans le programme. L'emploi de END en fin de programme est facultatif.

FONCTION **EOF**

Syntaxe : EOF([#]n)

Exemple :
```
OPEN "O",#1,"DAT"
PRINT #1,"123456"
CLOSE #1
OPEN "I",#1,"DAT"
DO
  PRINT INP(#1),EOF(#1)
  EXIT IF EOF(#1)
LOOP
```

Fonction : Indique si l'on se trouve à la fin du fichier ayant pour numéro de canal n.

Explication :

'*n*' est une expression entière comprise entre 0 et 99 qui correspond au numéro d'un canal de données ouvert avec OPEN.

Pour chaque canal de données, il existe un pointeur de fichier (pointeur de lecture et d'écriture) qui pointe sur un octet précis du fichier.

Si le pointeur est en fin de fichier, la fonction EOF retourne la valeur -1 (cette valeur correspond à la valeur logique 'vrai').

Dans les autres cas, la fonction retourne la valeur 0.

La ligne de programme EXIT IF EOF(#1) tirée de l'exemple évite le message d'erreur 'Fin de fichier atteinte, EOF'.

Cette instruction ne peut être utilisée que pour des fichiers sur disquette (et non pour CON: LST: etc...).

INSTRUCTION **ERASE**

Syntaxe : ERASE tableau()

Abrev. : ER

Exemple :
```
DIM A(100,50,5)
PRINT FRE(0)
ERASE A()
PRINT FRE(0)
DIM A(3000)
```

Fonction : Efface un tableau et libère la place mémoire correspondante.

Explication :

L'instruction ERASE permet d'effacer un tableau de nom 'tableau' qui a été défini avec l'instruction DIM. Par la suite, il est naturellement possible de redéfinir un tableau de même nom.

Si un tableau n'est plus utilisé dans la suite du déroulement d'un programme, il peut être avantageux (surtout s'il reste peu de place disponible en mémoire principale) de l'effacer avec ERASE. De cette manière, la place qui a été réservée est à nouveau entièrement disponible.

FONCTION **ERR**

Syntaxe : ERR

Exemple :
```
ON ERROR GOSUB Routinerreur
PRINT SQR(-1)
PROCEDURE Routinerreur
  PRINT "Erreur No : ";ERR
RETURN
```

Fonction : Retourne le code de l'erreur rencontrée.

Explication :

ERR est en réalité une variable qui contient le code de la dernière erreur rencontrée (voir annexe B).

ERR ne peut pas etre utilisé comme nom de variable, même en relation avec LET.

(voir aussi FATAL, ERROR et ON ERROR).

INSTRUCTION ERROR

Syntaxe : ERROR n

Abrev. : ERR

Exemple :
```
PRINT "Tapez le code d'erreur : ";
INPUT f
ERROR f
```

Fonction : Simule l'apparition d'une erreur dont le code est n.

Explication :

n est une expression entière dont la valeur est comprise entre -128 et 127.

ERROR n simule une erreur dont le code d'erreur est n. Si l'instruction ON ERROR GOSUB a été utilisée, on se branche au sous-programme correspondant ; sinon un message d'erreur relatif au code n est affiché (voir annexe B) et le programme est interrompu.

FONCTION **EVEN**

Syntaxe : EVEN(n)

Exemple :
```
REPEAT
  INPUT "Entrez un nombre pair, s.v.p : ";N
UNTIL EVEN(N)
```

Fonction : Détermine la parité d'un nombre.

Explication :

'*n*' est une expression numérique. Si n est pair, EVEN(n) retourne -1. Si n est impair, on obtient 0.

(voir ODD).

INSTRUCTION/FONCTION EXEC

Syntaxe : EXEC flg,nom,cmd,env
EXEC(flg,nom,cmd,env)

Exemple :
```
RESERVE FRE(0)-10000
EXEC 0,"TEST.PRG","","
RESERVE FRE(0)+10000-255
```

Fonction : Chargement et exécution de programmes machines ou de programmes compilés, à partir de la disquette.

Explication :

'*flg*' peut prendre deux valeurs : 0=Load and Go, 3=Load Only.

'*nom*' est le nom du programme.

'*cmd*' est la ligne de commande à transmettre (voir BASEPAGE),

'*env*' est la chaîne-environnement (bien souvent, "" suffit).

Le programme en question est chargé en mémoire à partir de la disquette, les adresses absolues sont transformées en adresses relatives, une page de base est créée - avec la ligne de commande - et, le cas échéant, le programme est exécuté. Dans l'exemple, on réduit tout d'abord la place mémoire réservée au BASIC de 10000 octets. Cette zone sera à nouveau allouée au BASIC à la fin de l'opération. Si EXEC est utilisé comme une fonction, on obtient soit le paramètre retourné par le programme, soit (pour flg=3) l'adresse de la page de base du programme chargé en mémoire.

(voir aussi RESERVE et HIMEM).

FONCTION **EXIST**

Syntaxe : EXIST("specfich")

Exemple :
```
OPEN "O",#1,"FICHIER.DOC"
CLOSE #1
PRINT EXIST("FICHIER")
PRINT EXIST("FIC*.DOC")
PRINT EXIST("*.*")
```

Fonction : Teste l'existence d'un fichier sur disquette.

Explication :

'specfich' est un nom de fichier qui peut posséder toutes les spécifications possibles (voir DIR).

Si le fichier existe, la valeur -1 est retournée. Si le fichier n'existe pas, la valeur 0 est retournée.

Un fichier n'est trouvé que si le nom qui a été entré coïncide exactement avec le nom dans le répertoire (extension inclue).

INSTRUCTION DE STRUCTURE EXIT

Syntaxe : EXIT IF Condition

Abrev. : E

Exemple :

```
DO
  A=A+1
  PRINT A
  EXIT IF A=20
LOOP
```

Fonction : Permet de sortir d'une boucle.

Explication :

Si l'instruction EXIT est rencontrée à l'intérieur d'une boucle, le déroulement du programme se poursuit à la première instruction en dehors de la boucle. Cette instruction permet de sortir de n'importe quelle boucle :

FOR...NEXT, DO...LOOP, REPEAT...UNTIL et WHILE...WEND.

FONCTION EXP

Syntaxe : EXP(x)

Exemples :
```
PRINT EXP(1)
PRINT EXP(0.5)
PRINT EXP(-2)
```

Fonction : Calcule l'exponentielle de x (e^x e=2.7182818285).

Explication :

'*x*' est une expression numérique quelconque.

La fonction EXP(x) calcule e (base du logarithme népérien) à la puissance x.

CONSTANTE FALSE

Syntaxe : FALSE

Exemple : Flag!=FALSE

Fonction : Constante 0

Explication :

Il s'agit simplement d'une autre écriture de la valeur booléenne "faux". FALSE est équivalent à 0.

(voir aussi TRUE).

FONCTION **FATAL**

Syntaxe : FATAL

Exemple :
```
ON ERROR GOSUB Routinerreur
ERROR 5
PROCEDURE Routinerreur
  PRINT "FATAL=";FATAL
RETURN
```

Fonction : Retourne la valeur 0 ou -1 selon le type de l'erreur.

Explication :

La fonction renvoie la valeur 0 pour les erreurs 'normales'. La valeur -1 est retournée pour toutes les erreurs qui ne permettent pas de retrouver l'adresse de la dernière instruction (en général, cela concerne uniquement les erreurs à l'intérieur du système d'exploitation qui conduisent à l'affichage bien connu des bombes et à la destruction du programme).

INSTRUCTION I/O — FIELD

Syntaxe : FIELD [#]n,expres AS varch...

Abrev. : FIE

Exemple :

```
OPEN "R",#1,"TEST",50
FIELD #1,20 AS A$,30 AS B$
FOR I=1 TO 4
  INPUT "Prénom,nom: ",P$,N$
  LSET A$=P$
  LSET B$=N$
  PUT #1,I
NEXT I
FOR I=4 DOWNTO 1
  GET #1,I
  PRINT A$,B$
NEXT I
```

Fonction : Subdivise des enregistrements en champs de données.

Explication :

'n' est une expression entière comprise entre 0 et 99 qui est le numéro d'un canal de données préalablement ouvert avec OPEN.
'expres' est une expression entière qui définit la longueur du champ.
'varch' est une variable chaîne de caractères (et non pas une variable tableau) qui reçoit les données.
L'ensemble *'expres AS varch'* peut être répété plusieurs fois (en les séparant par des virgules) si l'enregistrement doit être subdivisé dans plusieurs champs.
La somme des longueurs des champs devrait être égale à la longueur de l' enregistrement.

A chaque canal de données ouvert, on ne peut faire correspondre qu'une seule instruction FIELD. ==>

INSTRUCTION I/O **FIELD suite**

On ne peut en aucun cas modifier la longueur de '*varch*', si cette variable doit encore être utilisée par les instructions GET et PUT. L'affectation des variables 'varch' devrait être réalisée avec les instructions LSET et RSET.

Dans l'exemple, on crée un fichier à accès direct qui est lu en partant de la fin.

Le BASIC GfA n'utilise pas de zone tampon pour ce mode d'accès. C'est pourquoi VARPTR(#n) n'est pas possible.

INSTRUCTION I/O — FILES

Syntaxe : FILES ["specfich" [TO "fich"]]

Abrev. : FILE

Exemples : FILES "A:*.*" TO "A:TABMAT"
FILES "A:*.*" TO "LST:"

Fonction : Liste les fichiers d'une disquette.

Explication :

'specfich' est un nom de fichier pour lequel toutes les spécifications sont possibles (voir DIR).

L'instruction FILES est presque identique à DIR, mais la liste obtenue est plus détaillée : à côté du nom de fichier, figurent également sa taille, l'heure et la date de création.

INSTRUCTION FILESELECT

Syntaxe : FILESELECT "specfich","nomfich",x$

Abrev. : FILESE

Exemple :

```
DO
  FILESELECT "\*.*",B$,A$
  EXIT IF A$=""
  PRINT A$
LOOP
```

Fonction : Met en place un boîte de selection de fichiers.

Explication :

'*specfich*' est appelé chemin de sélection. Il s'agit du critère de séléction des fichiers pour lequel toutes les spécifications sont possibles (voir DIR). Le critère de séléction minimal est "*.*". Dans ce cas, tous les fichiers de la directory principale sur l'unité utilisée sont affichés dans la boîte de séléction.

'*x$*' est une variable chaîne de caractères quelconque à laquelle est affecté le nom du fichier choisi par l'utilisateur. En cliquant sur annulation, on affecte à x$ la chaîne vide.

Pour une meilleure compréhension, il est vivement conseillé de tester l'exemple.

INSTRUCTION GRAPHIQUE FILL

Syntaxe : FILL x,y

Abrev. : FI

Exemple :

```
BOX 10,10,100,100
DRAW 200,20 TO 300,20 TO 300,300...
        . . . TO 200,300 TO 200,22
DEFFILL 1,2,9
FILL 50,50
DEFFILL 1,2,8
FILL 250,280
```

Fonction : Remplit une surface avec différents motifs.

Explication :

x et y sont les coordonnées du point à partir duquel le remplissage commence.

L'origine des coordonnées se situe dans le coin supérieur gauche de l'écran. Le point central de l'écran a pour coordonnées (320,200) en haute résolution. Si le remplissage doit se restreindre à une surface limitée, il faut faire attention à ce que la frontière de cette surface ne comporte aucun trou (voir exemple).

Le motif de remplissage est défini avec l'instruction DEFFILL.

FONCTION FIX

Syntaxe : FIX(x)

Exemples :
```
A=3.1415
PRINT FIX(A)
PRINT FIX(-12)
PRINT FIX(-1.99)
```

Fonction : Retourne la partie de x située avant la virgule.

Explication :

'*x*' est une expression numérique quelconque.

Pour des valeurs positives de x, cette fonction est identique à INT. La différence apparait pour les nombres négatifs. Ainsi, FIX(-1.99) donne le résultat -1, alors que INT(-1.99) donne -2. FIX (et non pas INT) est complémentaire à FRAC (voir fonction FRAC). On a :

x = FIX(x) + FRAC(x)

La fonction FIX est identique à la fonction TRUNC.

INSTRUCTION DE STRUCTURE FOR ... NEXT

Syntaxe :
```
FOR Var = d [DOWN]TO f [STEP s]
NEXT Var
```

Abrev. :
```
F [DOWN]TO  STEP
N
```

Exemple :
```
FOR I=3 to 5
  FOR J=5 DOWNTO 3
    FOR K=1 TO 5 STEP 2
      PRINT I,J,K
    NEXT K
  NEXT J
NEXT I
```

Fonction : Met en place une boucle dont le nombre de passages est fixé.

Explication :

On affecte à la variable Var la valeur d et on exécute les instructions entre FOR et NEXT. Var est ensuite (une fois sur NEXT) augmenté de la valeur s et on teste si le contenu de Var est supérieur à f. Si ce n'est pas le cas, le programme se rebranche sur FOR. Ce processus est répété jusqu'à ce que Var dépasse la valeur de fin f.

Si s est négatif, le processus est inversé : Var est diminué, et on teste si Var est plus petit que f.

Si la partie facultative STEP s est absente, s prend la valeur 1 par défaut.

Si on utilise DOWNTO à la place de TO, il est interdit d'employer STEP s et s prend toujours la valeur -1. ==>

INSTRUCTION DE STRUCTURE FOR ... NEXT
suite

Les boucles FOR-NEXT peuvent être emboîtées les unes dans les autres.

La vitesse d'exécution d'une boucle FOR-NEXT peut être sensiblement augmentée si on choisit une variable entière pour le compteur de boucle 'Var'. L'exemple suivant met en évidence une diminution du temps d'exécution qui passe de 1.19 s à 0.40 s par la simple utilisation de la variable entière I% à la place de I :

```
T=TIMER
FOR I=1 TO 10000
NEXT I
PRINT (TIMER-T)/200
T=TIMER
FOR I%=1 TO 10000
NEXT I%
PRINT (TIMER-T)/200
```

INSTRUCTION I/O — FORM INPUT

Syntaxe : FORM INPUT n,var

Abrev. : F MINPUT

Exemple :

```
PRINT "Entrez votre nom"
PRINT "(max. 15 lettres)"
PRINT AT(15,15);
FORM INPUT 15,Nom$
PRINT "Vous vous appelez ";Nom$
```

Fonction : Permet de saisir une chaîne de caractères dont la taille est limitée, durant l'exécution du programme.

Explication :

'*n*' indique la longueur maximale de la chaîne de caractères.

'*var*' est le nom de cette chaîne.

Si l'interpréteur rencontre l'instruction FORM INPUT, il interrompt l'exécution du programme et l'utilisateur a la possibilité d'entrer au clavier une chaîne de caractères. Le nombre maximum de caractères pouvant être introduits est fixé par n. Si on atteint cette taille limite, un son de cloche retentit.

Une chaîne de caractères saisie au clavier comporte au maximum 255 caractères.

==>

INSTRUCTION I/O **FORM INPUT**
suite

Les caractères spéciaux peuvent être saisis au clavier et affichés de trois manières différentes :

- en appuyant simultanément sur la touche <ALTERNATE> et une autre touche.

- en appuyant simulanément sur les touches <CONTROL> et <S>, puis sur une touche quelconque.

- en appuyant simultanément sur les touches <CONTROL> et <A>, puis en tapant le code ASCII du caractère à afficher.

Lors d'une saisie au clavier, on peut comme d'habitude effectuer des corrections à l'aide des touches <DELETE> et <BACKSPACE>. Avec les touches de déplacement 'à droite' et 'à gauche', on peut atteindre n'importe quel caractère déjà introduit. Avec les touches de déplacement 'vers le haut' et 'vers le bas', on peut placer le curseur respectivement au début et à la fin de la chaîne introduite.

Ce mode d'introduction des données est le même pour INPUT et LINE INPUT.

INSTRUCTION I/O **FORM INPUT AS**

Syntaxe : FORM INPUT n AS var

Abrev. : F MINPUT

Exemple :
```
PRINT "Modification de la chaîne"
LET Chaine$="Ancienne Chaîne"
PRINT AT(15,15);
FORM INPUT 15 AS Chaine$
```

Fonction : Permet de modifier une chaîne de caractères durant l'exécution du programme.

Explication :

'*n*' indique la longueur maximale de la chaîne devant être saisie.

'*var*' est le nom d'une variable alphanumérique. Cette instruction est similaire à FORM INPUT ; la seule différence réside dans le fait que l'ancien contenu de la chaîne de caractères 'var' est affiché, avant d'être modifié.

FONCTION FRAC

Syntaxe : FRAC(x)

Exemples :
```
A=-10.1234
PRINT FRAC(3.1415)
PRINT FRAC(A)
PRINT TRUNC(A)+FRAC(A)
```

Effet : Retourne la partie de x située après la virgule.

Explication :

'x' est une expression numérique quelconque.

FRAC(x) tronque la partie de x avant la virgule et restitue la partie de x après la virgule. Pour des valeurs entières de x, FRAC(x) retourne donc la valeur 0.

On a : FRAC(x)=x-TRUNC(x)

FRAC est la fonction complémentaire de TRUNC et non de INT (voir INT et TRUNC).

FONCTION FRE

Syntaxe : FRE(x)

Exemple :
```
PRINT FRE(A%)
DIM N$(1000)
PRINT FRE(0)
```

Fonction : Calcule la taille en octets de la place mémoire encore disponible.

Explication :

Le paramètre x, qui peut être une expression numérique, n'est pas pris en compte.

A l'appel de la fonction FRE, le BASIC GfA calcule, en nombre d'octets, la place encore disponible de la mémoire principale après avoir effectué une 'garbage-collection' (collecte et effacement de la place non utilisée de la mémoire principale).

INSTRUCTION FULLW

Syntaxe : FULLW n

Abrev. : FU

Exemple : FULLW 2

Fonction : Agrandit la fenêtre numéro n à la taille de l'écran.

Explication :

'*n*' est une expression numérique qui indique le numéro de la fenêtre.

Si la fenêtre n n'est pas encore ouverte, son ouverture a lieu automatiquement.

INSTRUCTION GET

Syntaxe : GET x0,y0,x1,y1,a$

Abrev. : GET

Exemple :

```
FOR I=1 TO 5
  CIRCLE 320,200,I*40
NEXT I
GET 0,0,320,399,A$
PUT 320,0,A$,3
```

Fonction : Affecte une zone rectangulaire de l'écran transformée en une suite de bits à la chaîne de caractères a$.

Explication :

(x0,y0) et (x1,y1) sont les deux sommets diagonalement opposés du rectangle qui doit être lu.

'a$' est une variable chaîne de caractères qui va recevoir la série de bits correspondante.

Avec l'instruction PUT, on peut inversement afficher une partie de l'écran préalablement mémorisée dans a$ à une place quelconque de l'écran (voir PUT). ==>

INSTRUCTION DE STRUCTURE GOSUB

Syntaxe : GOSUB nom [(listexpres)]

Abrev. : GO ou @

Exemple :

```
PRINT "Programme principal"
GOSUB 1er.niveau
PRINT "Retour du 1"
PROCEDURE 1er.niveau
  PRINT "Procedure 1"
  GOSUB 2eme.niveau(3,2)
  PRINT "Retour du 2"
  PRINT A,B
RETURN
PROCEDURE 2eme.niveau(A,B)
  PRINT A,B
  PRINT "Procedure 2"
RETURN
```

Fonction : Appelle la procédure dont le nom est 'nom'.

Explication :

'nom' est le nom de la procédure appelée.

Le nom d'une procédure peut commencer par un chiffre, à l'inverse des noms de variables. Pour les caractères suivants, le BASIC GfA autorise : les lettres, les chiffres, le souligné et le point.

'listexpres' est une liste d'expressions séparées par des virgules qui sont transmises aux variables locales de la procédure (voir PROCEDURE).

Lorsque l'interpréteur rencontre une instruction GOSUB, il se branche sur la procédure avec le nom correspondant. ==>

INSTRUCTION DE STRUCTURE **GOSUB**
suite

Il est possible, à l'intérieur d'une procédure, d'appeler une autre procédure par GOSUB.

Il est même possible, à l'intérieur d'une procédure, d'appeler à nouveau la procédure où l'on se trouve (appel récursif).

L'exemple appelle, à partir du programme principal, la procédure '1er.niveau'. A l'intérieur de la procédure '1er.niveau', on transmet, par l'appel de la procédure '2eme.niveau', les valeurs 3 et 2 aux variables A et B. Enfin, la procédure '2eme.niveau' affiche ces deux variables à l'écran. Pour bien montrer que A et B sont locales à la deuxième procédure, on affiche à nouveau les variables A et B après avoir quitté la deuxième procédure : on obtient deux fois la valeur 0.

(voir PROCEDURE, RETURN, LOCAL).

INSTRUCTION DE STRUCTURE GOTO

Syntaxe : GOTO etiquette

Abrev. : GOT

Exemple :
```
Boucle:
PRINT "Boucle sans fin ";
GOTO Boucle
```

Fonction : Permet un branchement inconditionnel.

Explication :

'etiquette' est une chaîne de caractères formée de lettres, de chiffres, du souligné et du point. Cette chaine peut commencer par un chiffre, à l'inverse des noms de variables.

Comme le BASIC GfA travaille sans les numéros de lignes, il est nécessaire de marquer l'endroit où doit se positionner le compteur de programme. On utilise pour cela un nom d'étiquette (label) suivi d'un double-point.

Lors d'une exécution, quand l'interpréteur rencontre un GOTO, il se branche de manière inconditionnelle après la ligne repérée par l'étiquette correspondante.

L'exemple simule une boucle DO...LOOP.

INSTRUCTION GRAPHIQUE GRAPHMODE

Syntaxe : GRAPHMODE n

Abrev. : G

Exemple :

```
DEFFILL 1,3,4
PBOX 10,100,150,250
GRAPHMODE 4
DEFFILL 1,3,5
PBOX 120,200,200,300
GRAPHMODE 1
PBOX 120,180,200,50
```

Fonction : Met en place le mode graphique :
n=1 : replace
n=2 : transparent
n=3 : xor
n=4 : reverse transparent

Explication :

Le mode graphique n'intervient que lorsque plusieurs images doivent se superposer.

Si n=1 (mode normal), l'ancienne image est remplacé par la nouvelle. Si n=2, la nouvelle image est transparente, c'est-à-dire que l'ancienne image apparait au travers de la nouvelle.

Pour n=4, c'est à peu près la même chose. La seule différence est que l'image affichée en dernier est représentée en inverse-vidéo.

Si n=3, chaque point de la nouvelle image est allumé, s'il était précédemment éteint et est éteint s'il était allumé. Par double affichage, ce mode graphique permet de créer des images clignotantes et mêmes des images en mouvement. ==>

INSTRUCTION GRAPHIQUE **GRAPHMODE**
suite

Ainsi, l'exemple simple suivant trace un cercle qui peut être déplacé sur l'écran à l'aide de la souris.

Si la touche gauche de la souris est enfoncée, une copie du cercle est fixée :

```
DO
  IF K=1 THEN
    GRAPHMODE 1
  ELSE
    GRAPHMODE 3
  ENDIF
  MOUSE X,Y,K
  CIRCLE X,Y,50
  CIRCLE X,Y,50
LOOP
```

INSTRUCTION I/O HARDCOPY

Syntaxe : HARDCOPY

Abrev. : H

Exemple :

```
GRAPHMODE 3
FOR I=0 TO 800 STEP 8
  BOX I MOD 640, I MOD 400, 639-I...
          ...MOD 640,399-I MOD 400
NEXT I
HARDCOPY
```

Fonction : Provoque une hardcopy.

Explication :

Une hardcopy est la sortie du contenu de l'écran sur imprimante (l'action simultanée sur les touches <ALTERNATE> et <HELP> provoque également une hardcopy).

L'exécution de la hardcopy peut être interrompue en appuyant sur les touches <ALTERNATE>-<HELP>. Si l'imprimante n'est pas branchée, le programme reprend la main au bout de 30 secondes environ.

FONCTION HEX$

Syntaxe : HEX$(x)

Exemple :
```
A=-1
B=&O22
PRINT HEX$(A)
PRINT HEX$(234)
PRINT HEX$(B)
```

Fonction : Convertit x en une chaîne de caractères représentant l'écriture héxadécimale de x.

Explication :

'x' est un entier relatif (entre -2147483648 et +2147483647) dans une représentation quelconque (c'est-à-dire la représentation décimale normale (sans préfixe), la représentation hexadécimale (préfixe & ou &H), la représentation octale (préfixe &O) ou la représentation binaire (préfixe &X)) ou une variable du type correspondant.

HEX$(x) convertit la valeur x en une chaîne de caractères représentant l'écriture héxadécimale de x (cela signifie que la variable B de l'exemple est convertie en la chaîne "12").

HEX$, de même que BIN$ et OCT$, considèrent les entiers relatifs comme une série de 32 bits, et les représentent donc sans signe, à l'inverse de la représentation décimale normale.

(voir aussi OCT$(x), BIN$(x) et STR$(x)).

INSTRUCTION GRAPHIQUE HIDEM

Syntaxe : HIDEM

Abrev. : HI

Exemple :
```
HIDEM
DO
  PLOT MOUSEX,MOUSEY
LOOP
```

Fonction : Fait disparaître le curseur de la souris.

Explication :

Il est parfois intéressant de faire disparaître totalement de l'écran le curseur de la souris. Cette instruction interrompt la représentation du curseur de la souris. Si par la suite on fait appel à l'AES (GEMSYS, ALERT, etc), la souris redevient visible. L'exemple ci-dessus permet de dessiner avec une souris invisible.

(voir aussi SHOWM).

FONCTION HIMEM

Syntaxe : HIMEM

Exemple : (voir EXEC)

Fonction : Retourne l'adresse de fin de la zone mémoire réservée au BASIC GfA.

Explication :

Le BASIC GfA occupe la zone mémoire située entre BASEPAGE et HIMEM (cette zone contient l'interpréteur lui-même, le programme, les variables, l'éditeur, etc...). Avec HIMEM, il est possible de déterminer la première adresse qui n'est plus réservée au BASIC GfA (normalement, cette adresse est située 16384 octets en dessous de la mémoire-écran). Au-delà de HIMEM, il devrait toujours rester 4000 octets de libre : cette place mémoire est nécessaire pour les boîtes de sélection de fichiers et pour d'autres appels au GEM (malheureusement, la taille exacte de cette zone mémoire n'est pas précisée dans la documentation fournie par Atari. Des tests ont montré qu'environs 2500 octets sont nécessaires, de telle sorte que 4000 octets devraient suffire dans tous les cas).

(voir aussi RESERVE et EXEC).

INSTRUCTION DE STRUCTURE IF

Syntaxe :

```
IF condition [THEN]
  blocprogramme
[ELSE]
  [blocprogramme]
ENDIF
```

Abrev. :

```
I
EL
EN
```

Exemple :

```
INPUT A,B
IF A=3
  PRINT "Vous avez gagné..."
  IF B=3 THEN
    PRINT "...le gros lot !"
  ELSE
    PRINT "...le lot de consolation !"
  ENDIF
ELSE
  PRINT "Vous avez malheureusement perdu."
ENDIF
PRINT "A bientôt !"
```

Fonction : Branche le programme sur différents blocs-programmes, suivant la valeur logique (vrai ou faux) de 'condition'.

Explication :

Le déroulement d'un programme dépend souvent de certaines conditions. ==>

INSTRUCTION DE STRUCTURE **IF**
suite

D'où l'utilité de l'instruction IF.

Si la condition est réalisée, on exécute les instructions du bloc-programme qui se trouve entre IF et ELSE (ou bien entre IF et ENDIF, si ELSE est absent). Si la condition n'est pas remplie, on exécute le bloc-programme entre ELSE et ENDIF (si l'instruction facultative ELSE manque, on passe à la suite).

Dans tous les cas, après le traitement de l'instruction IF, le programme poursuit son exécution en se branchant sur la première instruction après ENDIF.

Comme cette instruction souvent utilisée possède une syntaxe différente de celle des langages BASIC classiques, il est fortement conseillé de tester le programme de l'exemple.

INSTRUCTION ARITHMETIQUE INC

Syntaxe : INC var

Abrev. : IN

Exemple :

```
T=TIMER
FOR I%=1 TO 10000
  INC A%
NEXT I%
PRINT (TIMER-T)/200
A%=0
T=TIMER
FOR I%=1 TO 10000
  A%=A%+1
NEXT I%
PRINT (TIMER-T)/200
```

Fonction : augmente var de la valeur 1.

Explication :

'*var*' doit être une variable numérique ou un élément d'un tableau numérique.

'*INC var*' est identique à 'var=var+1'.

L'avantage de l'instruction INC réside dans la concision de sa syntaxe (gain de place mémoire) et surtout dans sa vitesse d'exécution (testez l'exemple !). Ainsi, l'utilisation de l'instruction INC augmente de façon sensible la vitesse d'exécution déjà très élevée par ailleurs (environ du simple au double dans l'exemple !).

INSTRUCTION	INFOW

Syntaxe : INFOW n,"info"

Abrev. : INF

Exemple : INFOW 2,"information"

Fonction : Transmet à la fenêtre numéro n la (nouvelle) ligne d'information 'info'.

Explication :

'*n*' est une expression numérique qui indique le numéro de la fenêtre dont on veut modifier la ligne d'information.

La chaîne de caractères 'info' devient la nouvelle ligne d'information (la ligne située sous le titre).

Si 'info' est une chaîne vide, la ligne d'information est effacée.

Comme on ne peut pas, une fois la fenêtre ouverte, faire apparaître ou disparaître complètement la ligne d'information (GEM), il faut faire précéder l'instruction OPENW par l'instruction INFOW n,"info" si on souhaite ou non afficher une ligne d'information. Avec INFOW n,"" avant OPENW, il n'y a pas de ligne d'information. Une ligne d'information, si elle existe, peut être modifiée après OPENW.

FONCTION **INKEY$**

Syntaxe : INKEY$

Exemple :

```
DO
  REPEAT
    ZS=INKEY$
  UNTIL ZS<>""
  PRINT LEN(Z$)
  PRINT LEFT$(ZS);" ";ASC(Z$),
  PRINT RIGHT$(ZS);" ";ASC(RIGHT$(Z$))
LOOP
```

Fonction : Lit un caractère entré au clavier.

Explication :

Cette fonction retourne une chaîne de caractères composée de 0, 1 ou de 2 caractères.

Si, lors de la dernière scrutation du clavier avec INKEYS, aucune touche n'a été enfoncée, la chaîne obtenue est vide. Si une touche a été enfoncée et que celle-ci possède un code ASCII (voir table CHRS), la fonction retourne le caractère correspondant. Pour les autres touches (par ex. F1 ou <HELP>), on obtient une chaîne de deux caractères. Le premier caractère de cette chaîne a toujours le code ASCII 0. Le deuxième caractère peut être obtenu à l'aide de l'exemple (ex : F1 donnerait CHR$(0)+CHR$(59)). L'exemple suivant interrompt l'exécution du programme jusqu'à ce qu'une touche soit enfoncée :

```
PRINT "Appuyez sur une touche s.v.p "
REPEAT
UNTIL INKEY$<>""
PRINT "      Merci."
```

FONCTION **INP**

Syntaxe : INP(x)
INP(#n)

Exemple :
```
OPEN "O",#1,"DAT"
PRINT #1,"ABC"
CLOSE #1
OPEN "I",#1,"DAT"
PRINT INP(#1)
PRINT "Appuyer sur une touche!"
PRINT INP(2)
```

Fonction : Lit un octet sur un périphérique ou sur un fichier.

Explication :

'*x*' est une expression entière comprise entre 0 et 5 et dont la signification est la suivante :

0 pour LST:
1 pour AUX:
2 pour CON:
3 pour MID:
4 (ne sert pas)
5 pour CON:

La fonction INP(x) lit exactement un octet sur le périphérique correspondant à l'indice x. L'exécution du programme est interrompue jusqu'à ce qu'un octet soit disponible (c'est-à-dire pour n=2 par ex. : jusqu'à ce qu'une touche soit enfoncée).
Pour les touches n'ayant pas de code ASCII, INP(2) retourne le code SCAN + 128 (F1=187).
Pour x<0, on obtient l'état du périphérique (cf INP?).
INP(#n) lit un octet sur le fichier correspondant au canal de données n. Pour déterminer si une donnée est présente dans un canal, on utilise status=BIOS(1,n).

FONCTION I/O INP?

Syntaxe : INP?(n)

Exemple :
```
IF INP?(1)
  @Lecture
ENDIF
```

Fonction : Fournit l'état (prêt à émettre ou non) d'un périphérique.

Explication :

'*n*' est le numéro du périphérique (0=LST: 1=AUX: 2=CON: 3=MID:).

Cette fonction retourne la valeur 0 s'il n'y a pas d'octet prêt et -1 (ou une valeur différente de 0) sinon. Dans l'exemple, on appelle un sous-programme dès lors que le port série a collecté quelque chose.

(voir aussi OUT?).

INSTRUCTION I/O **INPUT**
INPUT #

Syntaxe : INPUT ["texte";(ou ,)]var[,var...]
INPUT #n,var[,var...]

Abrev. : INP

Exemples :
```
INPUT K
PRINT AT(15,15);
INPUT "Votre nom ";N$
INPUT "âge et domicile: ",A,D$
```

Effet : Permet d'effectuer une saisie au clavier pendant l'exécution du programme.

Explication :

'texte' est une chaîne de caractères quelconque qui est affichée sur l'écran avant la saisie des données. Ce texte doit toujours être donné entre guillemets.

'var' est un nom de variable quelconque.

Quand l'interpréteur rencontre une instruction INPUT, il interrompt l'exécution du programme et l'utilisateur a la possibilité d'entrer des données. Avec INPUT #n, les données sont lues sur le canal n. Si le texte (facultatif) est présent, il est affiché à l'écran et le curseur se positionne à droite du texte. Si un point-virgule sépare 'texte' et 'var', le texte est suivi d'un point d'interrogation et d'un blanc. Si texte' et 'var' sont séparés par une virgule, l'introduction des données commence immédiatement à droite du texte.

Si les données introduites ne sont pas du même type que les variables correspondantes, une sonnerie retentit au moment de la saisie ==>

et la saisie peut être réeffectuée. Si la saisie est effectuée à partir d'un fichier, un message d'erreur est affiché.

Une chaîne de caractères saisie avec INPUT ne peut dépasser 255 caractères. Lors d'une saisie au clavier, on peut introduire des caractères spéciaux de 3 façons différentes :

- en appuyant simultanément sur la touche <ALTERNATE> et une autre touche.
- en appuyant simultanément sur les touches <CONTROL> et <S>, puis sur une touche quelconque.
- en appuyant simultanément sur les touches <CONTROL> et <A>, puis en tapant le code ASCII correspondant au caractère spécial.

Lors d'une saisie, on peut comme toujours effectuer des corrections avec <DELETE> et <BACKSPACE>. Avec les touches de déplacement <à droite> et <à gauche>, on peut atteindre n'importe quel caractère déjà introduit. Avec les touches de déplacement <vers le haut> et <vers le bas>, on peut positionner le curseur respectivement au début et à la fin de la partie introduite.

Si les données saisies comportent une virgule, seule est prise en compte la partie avant la virgule (à l'inverse de LINE INPUT). Cependant, la virgule et la partie suivant la virgule à l'intérieur d'une chaîne de caractères sont prises en compte si la chaîne est comprise entre guillemets. Les guillemets, quant à eux, ne sont pas mémorisés.

Si plusieurs variables suivent l'instruction INPUT, les données introduites doivent être séparées par des virgules ou des <RETURN>.

FONCTION **INPUT$**

Syntaxe : INPUT$(x[,#n])

Exemple :
```
OPEN "O",#1,"DAT"
PRINT #1,"GfA-BASIC"
CLOSE #1
OPEN "I",#1,"DAT"
PRINT INPUT$(3,#1)
PRINT "Introduisez 5 caractères !"
PRINT INPUT$(5)
```

Fonction : lit x caractères à partir du clavier ou d'un fichier et retourne la chaîne ainsi obtenue.

Explication :

'*x*' est une expression entière, dont la valeur est comprise entre 0 et 32767 (longueur maximale d'une chaîne).

'*n*' est une expression entière entre 0 et 99 qui correspond au numéro d'un canal de données ouvert grâce à OPEN.

Si le terme facultatif est absent, la fonction INPUT$(x) lit x caractères à partir du clavier et retourne la chaîne de caractères correspondante.

En ajoutant le terme '#n', les x caractères qui doivent former la chaîne sont lus à partir du fichier relatif au canal n, puis sont transmis par la fonction.

FONCTION **INSTR**

Syntaxe : INSTR ([n,]a$,b$) ou
INSTR (a$,b$[,n])

Exemple :
```
N$="BASIC GfA"
S$="BASIC"
PRINT INSTR$(N$,"A")
PRINT INSTR$(4,N$,"A")
PRINT INSTR$("BASIC GfA","fB")
PRINT INSTR$(N$,S$)
```

Fonction : Recherche si la chaîne de caractères b$ est contenue dans a$, et en donne la position.

Explication :

'n' est une expression numérique qui indique à quelle position de a$ la recherche doit commencer. Si n est absent, la recherche commence à partir du premier caractère de a$.

'a$' et 'b$' sont des chaînes de caractères quelconques.

Si a$ contient b$, la fonction retourne la position dans a$ à laquelle commence la sous-chaine b$.

Si a$ ne contient pas b$, la valeur 0 est retournée.

Si a$ est une chaîne d'au moins un caractère et b$ une chaîne vide, on obtient également la valeur 0.

Si les deux chaînes a$ et b$ sont vides, on obtient la valeur 1.

FONCTION **INT**

Syntaxe : INT(x)

Exemple :
```
A=3.1415
PRINT INT(A)
PRINT INT(678)
PRINT INT(-1.001)
```

Fonction : Calcule la partie entière de x.

Explication :

'*x*' est une expression numérique quelconque.

La fonction INT(x) calcule le plus grand entier relatif qui est inférieur ou égal à x.

Pour x positif, cela revient à tronquer la partie de x située après la virgule (dans ce cas, INT est identique à la fonction TRUNC).

Pour x négatif, INT arrondit x dans le sens des x négatifs. Ainsi l'exemple INT(-1.001) retourne la valeur -2.

INSTRUCTION I/O **KILL**

Syntaxe : KILL "specfich"

Abrev. : K

Exemple :
```
OPEN "O",#1,"EXEMPLE"
CLOSE
FILES
KILL "EXEMPLE"
PRINT
FILES
```

Fonction : Détruit un fichier de la disquette.

Explication :

'specfich' est une expression permettant de sélectionner un ou plusieurs fichiers (voir DIR).

Cependant, l'instruction KILL détruit un seul fichier, à savoir le premier fichier répondant au critère de sélection de 'specfich'.

Les guillemets entourant 'specfich' sont placés automatiquement par le BASIC GfA, au cas où ils n'auraient pas été tapés.

FONCTION LEFT$

Syntaxe : LEFT$(string[,n])

Exemple :
```
N$="BASIC GfA"
PRINT LEFT$(N$)
PRINT LEFT$(N$,3)
PRINT LEFT$(N$,12)
```

Fonction : Prend le premier caractère ou les n premiers caractères (de gauche) de la chaîne 'string'.

Explication :

'*string*' est une chaîne de caractères ou une variable chaîne de caractères.

'*n*' est normalement un entier naturel ou une variable de même type (cependant, on peut entrer un nombre décimal. L'éditeur ne prendra en compte que sa partie entière).

Si le paramètre facultatif n est absent, la fonction retourne uniquement le premier caractère de la chaîne 'string'.

Autrement, la fonction retourne une sous-chaîne de 'string' qui est composée des n premiers caractères (caractères de gauche). Les caractères blancs sont également pris en compte.

Si n est supérieur au nombre de caractères de 'string', la fonction retourne la chaîne 'string' elle-même. Si n=0, on obtient la chaîne vide.

FONCTION **LEN**

Syntaxe : LEN(x$)

Exemple :
```
A$=" GfA"
PRINT LEN(A$)
PRINT LEN("BASIC"+A$)
```

Fonction : Calcule la longueur de la chaîne de caractères x$.

Explication :

'*x$*' est une chaîne de caractères quelconque.

LEN calcule le nombre de caractères de la chaîne x$. Les caractères non affichables et les caractères blancs sont également pris en compte. On obtient dans l'exemple les valeur 4 et 9.

INSTRUCTION LET

Syntaxe : [LET] var = expres

Abrev. : LE

Exemple :
```
DIM A(18)
LET A(15) = 2*993
LET M$ = "GfA"
B = A(15)
N$ = M$
PRINT N$;B
```

Fonction : Affecte la valeur d'une expression à une variable.

Explication :

'*var*' peut être une variable numérique, une variable alphanumérique ou un élément de tableau.

'*expres*' est soit une variable (de même type que 'var'), soit un nombre, une chaîne de caractères, une formule de calcul, soit encore une fonction.

Dans cette instruction, le terme 'LET' est optionnel. Ainsi, LET A = B est identique à A = B.

Il faut noter que le type de 'var' et le type de 'expres' doivent être identiques (si les types ne coïncident pas entre eux, le BASIC GfA le signale lors du contrôle de la syntaxe).

INSTRUCTION GRAPHIQUE LINE

Syntaxe : LINE x0,y0,x1,y1

Abrev. : LI

Exemple :
```
LINE 0,0,639,399
LINE 639,0,0,399
```

Fonction : Relie les points (x0,y0) et (x1,y1) par une droite.

Explication :

L'origine des axes est le coin supérieur gauche de l'écran. x0 et y0 sont les coordonnées du point de départ, x1 et y1 les coordonnées du point d'arrivée.

L'exemple ci-dessus trace les deux diagonales de l'écran.

Cette instruction est identique à l'instruction DRAW x0,y0 TO x1,y1.

(voir aussi DEFLINE).

INSTRUCTION I/O	LINE INPUT LINE INPUT #

Syntaxe : LINE INPUT ["texte";(,)]var[,var...]
LINE INPUT #n,var[,var...]

Abrev. : LI INPUT

Exemple :
```
LINE INPUT K$
LINE INPUT A$,B$,C$
PRINT AT(15,15);
LINE INPUT "Votre nom";N$
LINE INPUT "Ville et pays: ",V$,P$
```

Fonction : Permet la saisie d'une chaîne de caractères pendant l'exécution du programme.

Explication :

'texte' est une chaîne de caractères quelconque qui est affichée à l'écran avant l'introduction des données. Ce texte doit toujours être compris entre guillemets.

'var' est le nom d'une variable chaîne de caractères.

Quand l'interpréteur tombe sur l'instruction LINE INPUT, il interrompt l'exécution du programme et l'utilisateur a la possibilité d'entrer une chaîne de caractères. Avec l'instruction LINE INPUT #n, la lecture a lieu sur le canal de données numéro n. A l'inverse de ce qui se passe avec l'instruction INPUT, une virgule n'est pas considérée comme un séparateur : elle est prise en compte dans la chaîne lue. Ici, le seul séparateur possible est <RETURN>. Cette différence mise à part, l'instruction LINE INPUT travaille de la même façon que l'instruction INPUT (voir INPUT).

INSTRUCTION I/O LIST

Syntaxe : LIST "nomfich"

Abrev. : LIS

Exemple :
```
PRINT "Sauvegarder sous format ASCII"
LIST "PROGR"
PRINT "Listing sur l'écran"
LIST ""
```

Fonction : Sauvegarde le programme en mémoire principale sur disquette, ou liste le programme sur l'écran.

Explication :

'*nomfich*' est le nom de fichier sous lequel le programme doit être sauvegardé.

Ce nom peut être précédé de toutes les spécifications possibles liées à la hiérarchie des fichiers (\) (voir par ex. DIR).

Si 'nomfich' est la chaîne de caractères vide, le listing du programme est affiché à l'écran. Dans tous les autres cas, l'instruction LIST est identique à l'option SAVE,A du menu de l'éditeur.

Les programmes, qui sont insérés dans d'autres programmes à l'aide de l'instruction MERGE, doivent avoir été préalablement sauvegardés avec sous la forme d'un fichier ASCII avec LIST.

Si aucune extension n'est présente, le BASIC GfA ajoute automatiquement le suffixe .LST .

Les guillemets qui entourent 'nomfich' sont automatiquement mis en place, s'ils n'ont pas été entrés.

INSTRUCTION I/O LLIST

Syntaxe : LLIST

Abrev. : LL

Exemple :
```
PRINT "Listing sur imprimante"
LLIST
```

Fonction : Imprime un listing du programme qui se trouve en mémoire principale.

Explication :

La sortie du listing du programme sur imprimante ne peut être arrêtée qu'en débranchant l'imprimante. 30 secondes environ s'écoulent avant que le programme ne reprenne la main et que l'on puisse à nouveau y travailler.

INSTRUCTION I/O **LOAD**

Syntaxe : LOAD "specfich"

Abrev. : LOA

Exemple : SAVE "PROGRAMM"
LOAD "PRO*"

Fonction : Charge un programme contenu sur fichier dans la mémoire de travail.

Explication :

'specfich' est un nom de fichier qui peut contenir diverses spécifications (voir par ex. DIR).

Si le nom de fichier entré au clavier n'a pas d'extension, le BASIC GfA ajoute automatiquement le suffixe .BAS .

De même, si les guillemets entourant 'filespec' manquent, ils sont automatiquement mis en place.

FONCTION **LOC**

Syntaxe : LOC([#]n)

Exemple :
```
OPEN "O",#1,"DAT"
PRINT #1,"1234567"
SEEK #1,3
PRINT LOC(#1)
```

Effet : Retourne la valeur du pointeur du fichier correspondant au canal n.

Explication :

'*n*' est une expression entière entre 0 et 99 qui indique le numéro d'un canal de données ouvert avec l'instruction OPEN.

A chaque canal de données, il correspond un pointeur de fichier (pointeur d'écriture et de lecture), qui pointe sur un certain octet dans le fichier. La fonction LOC retourne la valeur de ce pointeur, c'est-à-dire la distance (calculée en nombre d'octets) du pointeur par rapport au début du fichier.

Cette instruction peut uniquement être utilisée pour les fichiers sur disquette (pas pour CON: LST: etc...).

INSTRUCTION DE STRUCTURE LOCAL

Syntaxe : LOCAL var[,var...]

Abrev. : LOC

Exemple :

```
A=1000
PRINT A
GOSUB Sousprg
PROCEDURE Sousprg
  LOCAL A
  I=I+1
  A=I
  PRINT A
  IF I=10 THEN
  ELSE
    GOSUB Sousprg
  ENDIF
  PRINT A
RETURN
```

Fonction : Déclare 'var' comme variable locale.

Explication :

'*var*' peut être de n'importe quel type, sauf un tableau. En BASIC GfA, on peut déclarer des variables locales à l'intérieur de procédures (sous-programmes).

Une variable située en-dehors de la procédure, n'est pas affectée par une modification du contenu de la variable locale, même si les deux variables ont le même nom.

L'exemple ci-dessus démontre bien cette propriété, en appelant 10 fois la même procédure de façon récursive.

(voir aussi GOSUB, PROCEDURE, RETURN).

FONCTION **LOF**

Syntaxe : LOF([#]n)

Exemple :
```
OPEN "O",#1,"DAT"
PRINT LOF(#1)
PRINT #1,"1234567"
PRINT LOF(#1)
```

Fonction : Calcule la taille du fichier correspondant au canal numéro n.

Explication :

'*n*' est une expression entière entre 0 et 99 qui correspond au numéro d'un canal de données ouvert à l'aide de l'instruction OPEN.

La fonction LOF(#n) retourne le nombre d'octets que contient le fichier correspondant au canal de données numéro n (taille du fichier).

Pour un fichier qui vient tout juste d'être ouvert avec OPEN "O", on obtient la valeur 0.

Cette instruction s'utilise uniquement pour des fichiers sur disquette (non pas pour CON: LST: etc...) .

FONCTION **LOG**
LOG10

Syntaxe : LOG(x)
LOG10(x)

Exemple :
```
A=2.7182818285
PRINT LOG(A)
PRINT LOG(A^2)
PRINT LOG10(10*10*10)
PRINT LOG10(2.456)
```

Fonction : Calcule le logarithme népérien (LOG) ou le logarithme à base 10 (LOG10).

Explication :

'x' doit être une expression numérique dont la valeur est supérieure à 0.

LOG(x) calcule le logarithme népérien de x (logarithme de base e=2.71828182...).

LOG10(x) calcule le logarithme décimal de x (logarithme de base 10).

FONCTION **LPOS**

Syntaxe : LPOS(n)

Exemple :

```
FOR I=1 TO 600
  LPRINT "A";
  IF LPOS(1)=30 THEN
    LPRINT
  ENDIF
NEXT I
```

Fonction : Indique la colonne où se trouve (dans le buffer de l'imprimante) la tête d'écriture de l'imprimante.

Explication :

'*n*' est un argument muet. Il peut être choisi arbitrairement. La fonction retourne la position horizontale de la tête d'écriture. La numérotation des colonnes commence ici à partir de la valeur 0.

La valeur obtenue ne correspond pas toujours avec la position physique réelle où se trouve la tête d'écriture, car ici comme pour TAB, on ne compte que les caractères qui sont affichés par l'imprimante. CR, LF et BS [CHR$(13, 10, 8)] sont l'objet d'un traitement spécial.

INSTRUCTION I/O **LPRINT**

Syntaxe : LPRINT [expressions[,][;]['']]

Abrev. : LPR

Exemple :
```
A$="GfA"
B=1986
LPRINT A$
LPRINT B
LPRINT A$,B;"GfA"
LPRINT A$;"""";B
LPRINT USING "###.##",PI*100
```

Fonction : Sort des données sur l'imprimante.

Explication :

'*expressions*' est un nombre quelconque d'expressions qui doivent être séparées par des virgules, des point-virgules ou des apostrophes. Si ces séparateurs manquent, un point-virgule est automatiquement mis en place.

Cette instruction travaille exactement de la même façon que l'instruction PRINT, à la seule différence qu'on ne peut pas utiliser le suffixe AT(x,y) pour une imprimante.

(voir instructions PRINT et PRINT USING).

INSTRUCTION LSET

Syntaxe : LSET var = string

Abrev. : LS

Exemple :

```
A$="AAAAAAAAA"
B$=SPACE$(7)
C$="GfA"
LSET A$=C$
LSET B$="BASIC GfA"
PRINT A$;B$
```

Fonction : Place la chaîne de caractères 'string' dans la chaîne de caractères 'var', en partant de la gauche.

Explication :

'*var*' est une variable alphanumérique. 'string' est une expression alphanumérique quelconque.

La chaîne de caractères 'string' est placée, en commençant par la gauche, dans la chaîne 'var'. Si la longueur de 'string' est plus petite que celle de 'var', la partie non utilisée de 'var' est remplie de blancs. Dans le cas contraire, la chaîne 'string' est tronquée à droite.

LSET est normalement utilisé en relation avec FIELD pour la création d'un fichier à accès direct. Dans ce cas, les données numériques doivent d'abord être converties en chaînes de caractères avec MKI$, MKL$, MKS$, MKF$ ou MKD$, avant de pouvoir leurs appliquer l'instruction LSET.

(Voir aussi LSET et RSET).

FONCTION **MAX**

Syntaxe : MAX(expres[,expres...])

Exemple :
```
a=17
b=3
a$="aaa"
PRINT MAX(a,b,2*2)
PRINT MAX(a$,"AAAA")
```

Fonction : Calcule la valeur maximale ou la chaîne la plus 'grande' de la liste d'expressions 'expres'.

Explication :

'expres' est une expression numérique ou alphanumérique quelconque. Toutes les expressions de la liste doivent appartenir à un et un seul type (numérique ou alphanumérique).

S'il s'agit d'une liste d'expressions numériques, la fonction calcule la valeur la plus grande.

Si ce sont des chaînes de caractères, MAX retourne la chaîne la plus 'grande'.

Cette dernière s'obtient en comparant les chaînes avec l'opérateur '>' (voir chapitre : opérateurs de comparaison).

La fonction inverse de MAX est MIN.

INSTRUCTION **MENU**

Syntaxe : MENU tableau()
MENU KILL
MENU OFF
MENU n,x

Abrev. : ME

Exemple : voir ON MENU GOSUB

Fonction : Permet la création et la modification de menus.

Explication :

'tableau' est un tableau de chaînes de caractères à une dimension qui contient le texte du menu.

Le tableau de chaînes de caractères doit être composé de la manière suivante : il faut introduire successivement le titre du menu et les différentes options de ce menu déroulant. Après chaque titre et le menu déroulant correspondant, doit figurer une chaîne vide. La dernière option du dernier menu doit être suivie de deux chaînes vides. Il est par ailleurs INDISPENSABLE que le premier menu déroulant soit construit comme suit : titre, ligne d'information, ligne séparatrice composée de tirets (signes moins), six chaînes arbitraires (longueur <>0) pour les Desk- accessories.

Avec l'instruction MENU tableau(), le menu est affiché et activé.

Avec *MENU KILL*, on "déconnecte" le menu, cependant que ce dernier n'est pas effacé.

Avec *MENU OFF*, les titres de menus éventuellement représentés en inverse vidéo, sont à nouveau affichés 'normalement'. ==>

INSTRUCTION **MENU suite**

Avec l'instruction MENU n,x , il est possible de modifier les lignes du menu.

'*n*' représente ici le numéro de la ligne du menu (ainsi que l'indice du tableau correspondant).

'*x*' est un nombre entier compris entre 0 et 3.

MENU n,0 efface le crochet situé devant la ligne de menu, qui a été placé avec MENU n,1.

MENU n,1 affiche un crochet devant la ligne de menu. Pour utiliser cette instruction à bon escient, il devrait toujours y avoir au moins un blanc au début de la ligne.

MENU n,2 représente une ligne de menu en écriture claire. Il est alors impossible de choisir l'option correspondante.

MENU n,3 représente une ligne de menu dans un écriture normale. L'option correspondante peut à nouveau être choisie.

Si une ligne de menu commence avec le signe 'moins', il est impossible de choisir cette option (comme avec MENU n,2). Si l'on veut déconnecter les desk- accessories, on remplace les six données muettes (dans l'exemple ON MENU GOSUB 1,2,3,4,5,6) par six signes 'moins' (-,-,-,-,-,-). On peut également 'déconnecter' et 'rebrancher' les desk-accessories avec MENU n,2 et MENU n,3.

(voir aussi ON MENU GOSUB et la fontion MENU).

FONCTION **MENU**

Syntaxe : MENU(n)

Exemple : voir ON MENU GOSUB

Fonction : La fonction MENU(n) permet de d'obtenir les paramètres utilisés dans l'instruction ON MENU GOSUB.

Explication :

Après ON MENU, la fonction fournit les valeurs qui ont été obtenues par l'appel de la fonction VDI 'event_multi' correspondante.

'n' est une expression entière.

Pour n=0, on obtient le numéro de l'option du menu déroulant qui a été (éventuellement) clickée.

Pour n compris entre 1 et 8, on obtient différents paramètres qui dépendent de l'évènement déclencheur.

Pour n=9, on obtient un flag qui indique quel évènement s'est produit (int_out[0]).

==>

FONCTION **MENU suite**

Pour n=10 et n=11, on obtient les coordonnées de la souris au moment du déclenchement (in_out[1 et 2]).

Pour n=12, on obtient l'état des touches de la souris (in_out[3]).

Pour n=13, on obtient l'état des touches de commutation (<SHIFT>, <ALT>, <CONTROL>) du clavier (in_out[4]).

Pour n=14, on obtient, dans l'octet de poids faible, le code Scan, et, dans l'octet de poids faible, le code ASCII de la touche activée. (in_out[5]).

Pour n=15, on obtient le nombre de click-souris (in_out[6]).

Pour n=-1, on obtient l'adresse de l'arbre objet du menu.

Dans la routine appelée par ON MENU KEY GOSUB, on peut interroger la touche de déclenchement avec :

```
IF MENU(14) AND 255
  KEY$=CHR$(MENU(14))
ELSE
  KEY$=MKI$(MENU(14))
ENDIF
```

KEY$ est alors indentique à la fonction classique INKEY$.

FONCTION **MID$**

Syntaxe : MID$(string,i[,n])

Exemple :
```
N$="BASIC GfA"
PRINT MID$(N$,5)
PRINT MID$(N$,1,3)
PRINT MID$(N$,0,3)
PRINT MID$(N$,3,12)
```

Fonction : Extrait une sous-chaîne de 'string' de n caractères à partir de la ième position.

Explication :

'*string*' est une chaîne de caractères (constante ou variable).

'*a*' *et* '*n*' sont normalement des entiers naturels (constantes ou variables). (Cependant, on peut également introduire des nombres décimaux. L'éditeur ne prendra en compte que la partie entière).

Si le paramètre facultatif **'n'** est absent, la fonction extrait tous les caractères de la chaîne 'string' à partir de la ième position.

Sinon, on obtient une sous-chaîne qui commence par le ième caractère et qui comprend n caractères (les caractères blancs sont aussi pris en compte).

Pour i=0, on obtient le même résultat que pour i=1.

Si n est supérieur au nombre de caractères de 'string' situés à droite du ième caractère (inclus), on obtient MID$(string,i).

Si n est nul, on obtient la chaîne vide. ==>

INSTRUCTION **MID$**

Syntaxe : MID$(varch,d[,n])=expch

Exemple :
```
A$="ABCDEFGHIJKL"
MID$(A$,5)="..."
MID$(A$,3,1)="QWER"
```

Fonction : Modification d'une partie d'une chaîne de caractères.

Explication :

'varch' est une variable alphanumérique,

'd' et 'n' des expressions entières,

'expch' une expression alphanumérique.

L'instruction MID$ permet de modifier la partie de la chaîne 'varch' commençant au d-ième caractère de 'varch' et comprenant n caractère (ou, si n est absent, autant de caractères que 'expch' en contient). La longueur de la chaîne 'varch' n'est en aucun cas modifiée. Avec l'exemple de programme ci-dessus, on obtient "ABQD...HIJKL".

(voir aussi RSET et LSET).

FONCTION **MIN**

Syntaxe : MIN(expres[,expres...])

Exemple :
```
a=17
b=3
a$="aaa"
PRINT MIN(a,b,2*2)
PRINT MIN(a$,"AAAA")
```

Fonction : Fournit la plus petite valeur numérique ou la plus 'petite' chaîne de caractères de la liste d'expressions 'expres'.

Explication :

'expres' est une expression numérique ou alphanumérique quelconque.

Toutes les expressions de la liste doivent appartenir à un même type (numérique ou alphanumérique).

S'il s'agit d'une liste d'expressions numériques, la fonction retourne la plus petite valeur.

Si ce sont des expressions alphanumériques, on obtient la plus 'petite' chaîne de caractères. Cette dernière est calculée en comparant les chaînes de caractères entre elles à l'aide de l'opérateur '<' (voir chapitre : opérateurs de comparaison).

La fonction inverse de MIN est MAX.

INSTRUCTION I/O MKDIR

Syntaxe : MKDIR "nomrep"

Abrev. : MK

Exemple :
```
MKDIR "A:\REP1"
MKDIR "A:\REP1\REP2"
FILES "A:\*.*"
FILES "A:\REP1\*.*"
```

Fonction : Crée de nouveaux répertoires.

Explication :

'*nomrep*' est le nom du répertoire que l'on veut créer. Les répertoires permettent de créer une hiérarchie des fichiers.

S'il y a uniquement un nom de répertoire entre les guillemets (sans utilisation de \), le nouveau répertoire fera partie du répertoire (directory principale ou autre répertoire) où l'on se trouve en ce moment (pour changer de répertoire : CHDIR).

Si on veut placer le nouvau répertoire dans un répertoire bien déterminé, il faut indiquer en plus le nom et le chemin souhaité. Ce dernier commence toujours par le caractère '\', si l'on part du répertoire principal.

Ainsi, l'instruction MKDIR "\KFZ\KAROS\TEILNR" crée un répertoire de nom TEILNR qui sera placé dans le répertoire KAROS du répertoire KFZ du répertoire principal.

Si l'on veut utiliser une unité de disquettes précise, il faut faire précéder le nom du répertoire par la lettre correspondant au lecteur voulu, suivie d'un double-point (ex : "B:Repert").

FONCTION	MKI$ MKL$ MKS$ MKF$ MKD$

Syntaxe :

```
MKI$(n)
MKL$(n)
MKS$(n)
MKF$(n)
MKD$(n)
```

Exemple :

```
A=0.1111
PRINT MKF$(A)
FOR I=0 TO 5
  Z=PEEK(VARPTR(A)+I)
  PRINT Z,CHR$(Z)
NEXT I
```

Fonction : Convertit une valeur numérique en une chaîne de caractères.

Explication :

'*n*' est une expression numérique. Les fonctions convertissent les valeurs de ces expressions numériques en chaînes de caractères, à savoir :

MKI$ convertit un entier sur 16 bits en une chaîne de deux caractères.

MKL$ convertit un entier sur 32 bits en une chaîne de quatre caractères.

MKS$ met 'n' sous le format compatible avec le BASIC-Atari (4 octets).

==>

FONCTION	MKI$ MKL$ MKS$ MKF$ MKD$ suite

MKF$ met 'n' sous le format spécifique au BASIC GfA (6 octets).

MKD$ met 'n' sous le format compatible avec le MBASIC (8 octets).

Toute donnée numérique que l'on veut stocker sur un fichier à accès direct doit d'abord être convertie en une chaîne de caractères à l'aide des fonctions précédentes.

L'exemple montre que le BASIC GfA stocke les nombres en mémoire interne sous un format de 6 octets, que l'on peut également obtenir en utilisant la fonction MKF$.

Les fonctions inverses respectives sont :

CVI, CVL, CVS, CVF et CVD.

INSTRUCTION **MONITOR**

Syntaxe : MONITOR [n]

Abrev. : MON

Exemple : (pas nécessaire).

Fonction : Appelle un moniteur résidant en mémoire ou une extension de commande.

Explication :

Cette instruction permet d'appeler un moniteur ou un debugger, si ce dernier a repositionné le vecteur Illegal-Instruction de telle sorte que le MSB (octet de poids fort) du vecteur soit nul avant le chargement du BASIC GfA (cela est par exemple réalisé par le debbuger d'Atari). Il est également possible de transmettre au registre D0.L le paramètre facultatif n, et de créer ainsi de nouvelles instructions (par ex. MONITOR *a$() pour trier un tableau de chaînes de caractères). En outre, on transmet au registre A0 l'adresse d'une table de pointeurs qui pointe sur la zone mémoire contenant les huit types de variables (voir TYPE). Ces huit pointeurs sont aussi utilisés par l'interpréteur.

INSTRUCTION GRAPHIQUE MOUSE

Syntaxe : *MOUSE x,y,z*

Abrev. : M

Exemple :

```
DO
MOUSE A,B,C
  PRINT AT(1,1);A,B,C
LOOP
```

Fonction : Fournit la position (x,y) de la souris et l'état k de ses touches.

Explication :

La souris peut être déplacée sur toute la surface de l'écran, c'est à dire qu'elle peut atteindre, en haute résolution, les points (0,0) jusqu'à (639,399).

L'origine des coordonnées se trouve dans le coin supérieur gauche de l'écran.

L'instruction MOUSE retourne la position instantanée de la souris et affecte la composante horizontale resp. verticale à x resp. y.

'*k*' contient l'état instantané des touches de la souris. Plus précisément : 0 si aucune touche n'est enfoncée, 1 si la touche gauche est enfoncée, 2 si la touche droite est enfoncée et 3 si les deux touches sont enfoncées.

FONCTIONS **MOUSEX**
MOUSEY
MOUSEK

Syntaxe :
```
MOUSEX
MOUSEY
MOUSEK
```

Exemple :
```
DO
  X=MOUSEX
  Y=MOUSEY
  K=MOUSEK
  PRINT AT(1,1);X,Y,K
LOOP
```

Fonction : Donnent la position de la souris et l'état de ses touches.

Explication :

En haute résolution, la position horizontale MOUSEX de la souris peut prendre une valeur comprise entre 0 et 639, sa position verticale une valeur comprise entre 0 et 399. La position (0,0) se trouve dans le coin supérieur gauche de l'écran.

MOUSEK indique l'état des touches de la souris. MOUSEK prend la valeur 1 ou 2 selon que la touche gauche ou la touche droite est enfoncée. Si les deux touches sont enfoncées, on obtient la valeur 3.

INSTRUCTION ARITHMETIQUE MUL

Syntaxe : MUL var,n

Abrev. : MU

Exemple :

```
DIM A%(10000)
ARRAYFILL A%(),5
T=TIMER
FOR I=1 TO 10000
  MUL A%(I),5
NEXT I
PRINT (TIMER-T)/200
T=TIMER
FOR I=1 TO 10000
  A%(I)=A%(I)*5
NEXT I
PRINT (TIMER-T)/200
```

Fonction : Multiplie le contenu de 'var' par n.

Explication :

'*var*' doit être une variable numérique ou un élément d'un tableau numérique.

'*n*' est un nombre ou également une variable numérique.

MUL var,n est identique à var=var*n. L'avantage de l'instruction MUL réside dans la vitesse d'exécution (testez l'exemple !). Ainsi, l'utilisation de l'instruction MUL augmente de façon sensible la vitesse d'exécution déjà très élevée par ailleurs (d'environ 30% dans notre exemple).

INSTRUCTION I/O — NAME

Syntaxe : NAME "ancienfich" AS "nouvfich"

Abrev. : NA

Exemple :
```
OPEN "O",#1,"OLD"
CLOSE #1
FILES
NAME "OLD" AS "NEW"
FILES
```

Fonction : Change le nom d'un fichier.

Explication :

'ancienfich' et 'nouvfich' sont des noms de fichiers. Le nom de fichier peut être précédé par son chemin (le chemin est relatif à la hierarchie des fichiers (voir par ex. MKDIR)).

L'instruction recherche (éventuellement dans le répertoire indiqué) un fichier dont le nom est 'ancienfich' et lui donne le nouveau nom 'nouvfich'. Le contenu du fichier n'est pas modifié par cette opération.

L'instruction NAME ne peut être utilisée que pour une même unité de disquettes : si 'ancienfich' se trouve sur l'unité A (en faisant précéder le nom par A:), 'nouvfich' appartiendra aussi à l'unité A. Cependant, à l'intérieur d'une même unité de disquettes, 'nouvfich' peut faire partie de n'importe quel répertoire. Ainsi par exemple, l'instruction suivante est permise :

```
NAME "B:CALC" AS "\AUTO\CALCUL"
```

Ici, on change le nom du fichier CALC de la directory principale de l'unité B en "CALCUL", et on transfère ce fichier dans le répertoire AUTO (le fichier CALC n'existe plus).

INSTRUCTION **NEW**

Syntaxe : NEW

Abrev. : NEW

Exemple : PRINT "Ce programme va s'autodétruire !"
NEW

Fonction : Efface le programme qui se trouve en mémoire principale et toutes les variables.

Explication : (inutile).

FONCTION **OCT$**

Syntaxe : OCT$(x)

Exemple :
```
A=-1
B=&H22
PRINT OCT$(A)
PRINT OCT$(234)
PRINT OCT$(B)
```

Fonction : Convertit la valeur x en une chaîne de caractères qui est la représentation octale de x.

Explication :

'*x*' est un nombre entier compris entre -2147483648 et +2147483647 dans une représentation quelconque (à savoir l'écriture décimale normale (sans préfixe), l'écriture héxadécimale (préfixe & ou &H), la représentation octale (préfixe &O), la représentation binaire (préfixe &X)) ou une variable du même type. OCT$(x) convertit la valeur x en une chaîne de caractères qui est la représentation octale de x. (c'est-à-dire que la variable B de l'exemple est transformée en "42".)

OCT$, tout comme BIN$ et HEX$, considèrent les nombres comme une suite de 32 bits et les représentent sans signe, à l'inverse de la représentation décimale normale.

(voir aussi HEX$(x), BIN$(x) et STR$(x)).

FONCTION **ODD**

Syntaxe : ODD(n)

Exemple :
```
REPEAT
  INPUT "Entrez un nonbre pair SVP ";N
UNTIL NOT ODD(N)
```

Fonction : Permet d'obtenir la parité d'un nombre.

Explication :

'*n*' est une expression entière.

ODD(n) retourne -I si n est impaire, et 0 si n est pair.

(voir aussi EVEN).

INSTRUCTION DE STRUCTURE ON...GOSUB

Syntaxe : ON expres GOSUB listeproc

Abrev. : ON GOSUB

Exemple :

```
DO
  INPUT A
  ON A GOSUB PRO1,PRO2
LOOP
PROCEDURE PRO1
  PRINT "1<= A <2"
RETURN
PROCEDURE PRO2
  PRINT "2<= A <3"
RETURN
```

Fonction : Branche à telle ou telle procédure, suivant la valeur de 'expres'.

Explication :

'*expres*' est une expression numérique quelconque.

'*listeproc*' est une liste de noms de procédures séparés par des virgules.

Si la valeur de l'expression est supérieure ou égale à 1 et strictement inférieure à 2, on exécute la première procédure de la liste. Si la valeur de l'expression est supérieure ou égale à 2 et strictement inférieure à 3, on exécute la deuxième procédure de la liste. Etc... Pour les valeurs auxquelles ne correspond pas de procédure (par ex. 0 ou 3 dans notre exemple), aucun appel de procédure n'est effectué.

INSTRUCTION **ON BREAK**
ON BREAK CONT
ON BREAK GOSUB

Syntaxe :
ON BREAK
ON BREAK CONT
ON BREAK GOSUB nom

Abrev. : (pas d'abréviation).

Exemple :
```
ON BREAK GOSUB Commentaire
PRINT "Interrompez le programme"
REPEAT
UNTIL MOUSEK
ON BREAK
PROCEDURE Commentaire
  PRINT "Interruption impossible"
RETURN
```

Fonction : Rend les arrêts de programme possibles, impossibles, ou provoque, en cas d'interruption de programme, le branchement à la procédure 'nom'.

Explication :

'nom' est un nom de procédure. En temps normal, un programme peut être interrompu par l'action simultanée sur les touches <CONTROL>, <SHIFT> et <ALTERNATE>. L'instruction ON BREAK CONT supprime cette possibilité d'interrompre un programme, l'instruction ON BREAK le permet à nouveau.

L'instruction 'ON BREAK GOSUB nom' provoque l'exécution de la procédure 'nom', dès que les trois touches précédentes sont enfoncées.

INSTRUCTION **ON ERROR**
ON ERROR GOSUB

Syntaxe : ON ERROR
ON ERROR GOSUB nom

Abrev. : (pas d'abréviation).

Exemple :
```
ON ERROR GOSUB Routinerreur
PRINT SQR(-1)
PRINT 3/0
ON ERROR
PRINT SQR(-1)
PROCEDURE Routinerreur
  PRINT "Erreur No : ";ERR
  ON ERROR GOSUB Routinerreur
  RESUME NEXT
RETURN
```

Fonction : Provoque en cas d'erreur le branchement à la routine 'nom'.

Explication :

'nom' est un nom de procédure. Lorsque survient une erreur, il n'y a pas d'interruption de programme et de message d'erreur affiché, mais branchement à la procédure dont le nom est 'nom'.

L'instruction doit figurer avant l'apparition de l'erreur dans le programme, et n'entraine qu'un seul appel de la procédure. Si une nouvelle erreur survient, l'interpréteur affiche à nouveau le message d'erreur correspondant. Si on veut qu'il y ait branchement pour chaque erreur, il faut ajouter à l'intérieur de la routine d'erreur, avant le retour au programme principal, l'instruction 'ON ERROR GOSUB nom' afin de réactiver ce processus (voir exemple). Pour revenir en mode normal (affichage d'un message d'erreur sans branchement), on utilise l'instruction ON ERROR.

INSTRUCTION ON MENU BUTTON

Syntaxe : ON MENU BUTTON c,m,e GOSUB proc

Exemple :

```
ON MENU BUTTON 8,1,1 GOSUB button
DO
  ON MENU
LOOP
PROCEDURE button
  PRINT MENU(15)'   !Nombre de clicks
  PRINT MENU(10)'MENU(11)   !x/y
  IF MENU(15)=4       !Fin si
    END               !quatre clicks
  ENDIF
RETURN
```

Fonction : Scrutation des clicks-souris.

Explication :

'c', *'m'* *et* *'e'* sont des expressions entières,

'proc' est le nom de la procédure à appeler.

'c'=nombre maximal de (C)licks à enregistrer,
'm'=(M)asque (1:gauche, 2:droit, 3:les deux),
'e'=(E)tat (comme 'm')

Dans l'exemple, on attend au maximum 8 clicks. A chaque série de clicks, le nombre de clicks et la position de la souris sont affichés. Une série de 4 clicks entraine la fin du programme.

(voir aussi MENU, ON MENU).

INSTRUCTION ON MENU GOSUB

Syntaxe :

```
ON MENU GOSUB proc1
ON MENU KEY GOSUB proc2
ON MENU MESSAGE GOSUB proc3
ON MENU IBOX a,x,y,l,h GOSUB proc4
ON MENU OBOX a,x,y,l,h GOSUB proc5
ON MENU
```

Abrev. : (pas d'abréviation).

Exemple :

```
DIM Mnl$(50)
FOR I=0 TO 50
  READ Mnl$(I)
  EXIT IF Mnl$(I)="***"
NEXT I
Mnl$(I)=""
Mnl$(I+1)=""
DATA Desk,Testmenu
DATA ---------------------------------------
DATA 1,2,3,4,5,6,""
DATA Fichier,Ligne1,Ligne2,Ligne3,""
DATA Help,Aide 1,Aide  2,""
DATA Edit,Edit1,Edit2,Edit3,Quit
DATA ***
MENU Mnl$()
OPENW 0
ON MENU GOSUB menue
ON MENU IBOX 1,100,100,200,100 GOSUB inbox1
DO
  ON MENU
LOOP
PROCEDURE menue
  PRINT "Choix : ";
  PRINT Mnl$(MENU(0))
  IF Mnl$(MENU(0))="Quit"
END                                    ==>
```

INSTRUCTION **ON MENU GOSUB suite**

```
    ENDIF
  RETURN
PROCEDURE inbox1
  DEFMOUSE 4
  ON MENU OBOX 1,100,100,200,100 GOSUB outbox1
RETURN
PROCEDURE outbox
  DEFMOUSE 0
  ON MENU IBOX 1,100,100,200,100 GOSUB inbox1
RETURN
```

Fonction : ON MENU ... réalise le traitement de divers événements (choix de l'option, etc...).

Explication :

'proc1', ... *'proc5'* sont les noms des procédures qui doivent traiter les événements correspondants.
ON MENU GOSUB détermine la procédure pour le traitement du choix de l'option.
ON MENU KEY GOSUB procède de même pour le traitement du clavier.
ON MENU IBOX ... GOSUB et ON MENU OBOX ... GOSUB déterminent les procédures pour le traitement des déplacements de la souris par rapport à des rectangles fixés (entrée : IBOX, sortie : OBOX). En ce qui concerne ces derniers, *'a'* détermine lequel des deux rectangles possibles est utilisé, *'x' et 'y'* sont les coordonnées du sommet supérieur gauche, *'l' et 'h'* sont la largeur et la hauteur.
ON MENU MESSAGE GOSUB détermine la routine pour le traitement des informations du GEM.
ON MENU seul permet d'interroger ces différents événements. Cette instruction doit donc être fréquemment exécutée (par ex. à l'intérieur d'une boucle).

INSTRUCTION I/O OPEN

Syntaxe : OPEN "mode",[#]n,"nomfich"[,len]

Abrev. : O

Exemple :

```
OPEN "O",#1,"NOM"
PRINT #1,"BASIC GfA"
OPEN "I",#2,"NOM"
DO
  EXIT IF EOF(#2)
  PRINT INPUT$(1,#2)
LOOP
CLOSE
```

Fonction : Ouvre un canal de données ou un canal vers un fichier sur disquette.

Explication :

'mode' doit être compris entre guillemets, et est l'un des caractères O, I, A, U ou R.

"O" ouvre un fichier pour y écrire en le réinitialisant.
"I" ouvre un fichier pour le lire.
"A" permet d'ajouter des données à un fichier.

"U" permet d'accéder à un fichier à la fois en lecture et en écriture. Dans ce cas, le fichier doit déjà avoir été initialisé avec "O".

"R" concerne les fichiers à accès direct.

'n' est une expression entière comprise entre 0 et 99 qui détermine le numéro sous lequel on pourra accéder au fichier dans la suite du programme. ==>

INSTRUCTION I/O OPEN
suite

'nomfich' est un nom de fichier. Si l'on veut que le fichier appartienne à une unité de disquettes précise, par exemple l'unité A, on fera précéder le nom du fichier par le préfixe 'A:'.

On peut également mettre à profit la hiérarchie des fichiers. C'est ainsi que 'B:\DATE\AGE' représente le fichier 'AGE' du répertoire 'DATE' sur l'unité de disquettes 'B'.

'nomfich' peut aussi être :

"CON:" pour la console
"LST:" oder 'PRN:' pour l'imprimante
"AUX:" pour l'interface série
"MID:" pour la sortie MIDI
"VID:" pour la console en mode 'transparent' (les caractères de contrôle sont transmis et non exécutés).
"IKB:" pour un accès direct en écriture au 6301-Keybord-Processor (Attention!).

Quand *'nomfich'* représente un des périphériques ci-dessus, le mode d'accès (O I A U R) n'est pas pris en compte. C'est pourquoi, il suffit dans ce cas d'introduire la chaîne vide "".

'len' a uniquement un sens pour les fichiers à accès direct : 'len' détermine la longueur d'un enregistrement. Si ce paramètre manque, la longueur d'un enregistrement prise par défaut est de 128 octets.

INSTRUCTION OPENW

Syntaxe : OPENW n[,x,y]

Abrev. : O W

Exemple : OPENW 2,320,200

Fonction : ouvre la fenêtre numéro n.

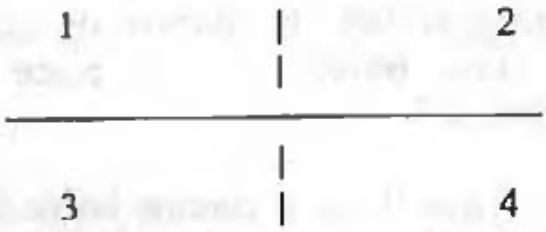
Explication :

'n' est une expression numérique, dont la valeur est le numéro de la fenêtre qui doit être ouverte.

Les paramètres facultatifs x et y sont les coordonnées du point de contact des quatre fenêtres possibles.

Les quatres fenêtres possibles sont définis comme les quatre cadrans d'un système de coordonnées, le point de contact correspondant à l'origine des axes.

La numérotation des fenêtres est la suivante :



Après OPENW 0, l'écran tout entier sans toutefois la ligne de menu est considéré comme une fenêtre. Avec OPENW 0,x,y , on définit l'origine des coordonnées pour les instructions graphiques.

INSTRUCTION **OPTION**
OPTION BASE 0
OPTION BASE 1

Syntaxe :
OPTION BASE 0
OPTION BASE 1
OPTION ["]TEXT["]

Abrev. :
OPT

Exemple :
```
OPTION "TRAPV +"
OPTION BASE 1
DIM A(9)
FOR i=1 TO 9
  A$(I)=STR$(I)
NEXT I
```

Fonction :
Modification de la base des tableaux ou gestion du compilateur.

Explication :

Avec OPTION BASE 0/1, il est possible de modifier la borne inférieure des dimensions d'un tableau (0 ou 1 selon l'instruction). Certains programmes, la plupart du temps des sous-programmes, peuvent ainsi économiser la place normalement réservée aux éléments d'indice 0.

Le BASIC GfA n'autorise que 0 ou 1 comme borne inférieure. Par ailleurs, OPTION BASE s'applique à tous les tableaux.

Avec OPTION "TEXT", on peut exécuter certaines instructions machines.

INSTRUCTION I/O OUT

Syntaxe : OUT x,a
OUT #n,a

Abrev. : OU

Exemple :
```
OPEN "O",#1,"DAT"
OUT #1,65
CLOSE #1
OPEN "I",#1,"DAT"
PRINT INPUTS(1,#1)
OUT 2,66
```

Fonction : Transmet un octet dont la valeur est 'a' à un périphérique ou à un fichier.

Explication :

'*x*' est une expression entière dont la valeur doit être comprise entre 0 et 5.
La signification de x est la suivante :

0 pour LST:
1 pour AUX:
2 pour CON:
3 pour MID:
4 pour IKB: (attention!)
et 5 pour VID:

'*a*' est une expression entière dont la valeur devrait être comprise entre 0 et 255. Si la valeur introduite est supérieure, on ne considère que a mod 256.
OUT x,a transmet un octet de valeur 'a' au périphérique repéré par x.
OUT #n,a transmet un octet de valeur 'a' au fichier correspondant au canal n.

FONCTION I/O OUT?

Syntaxe : OUT?(n)

Exemple :

```
DO
   OUT 0,0      !est ignoré
  EXIT IF OUT?(0)
  PRINT "Branchez l'imprimante!!!"
LOOP
```

Fonction : Permet d'obtenir l'état (prêt à recevoir ou non) d'un périphérique.

Explication :

'n' est le numéro du périphérique (0=LST:, 1=AUX:, 2=CON: (toujours prêt), 3=MID:).

La fonction *OUT?* retourne la valeur -1 (ou une valeur <> 0), si un caractère peut être envoyé au périférique. Dans l'exemple ci-dessus, on vérifie si l'imprimante est prête.

L'instruction *OUT 0,0* est nécessaire si l'imprimante n'a pas encore été utilisée ; elle est ignorée par (presque) toutes les imprimantes.

(voir INP?).

INSTRUCTION PAUSE

Syntaxe : PAUSE x

Abrev. : PA

Exemple :
```
PRINT "Patientez 5 secondes s.v.p !"
PAUSE 250
PRINT "Et voilà !"
```

Fonction : Interrompt l'exécution du programme pendant un certain temps.

Explication :

'*x*' est une expression numérique dont la valeur comprise entre -2147483648 et +2147483647.

L'instruction provoque une pause dans l'exécution du programme, dont la durée en seconde est donnée par x/50.

	INSTRUCTION GRAPHIQUE	**PBOX**
Syntaxe :	PBOX x0,y0,x1,y1	
Abrev. :	PB	

	INSTRUCTION GRAPHIQUE	**PCIRCLE**
Syntaxe :	PCIRCLE x,y,r[,phi0,phi1]	
Abrev. :	PC	

	INSTRUCTION GRAPHIQUE	**PELLIPSE**
Syntaxe :	PELLIPSE x,y,rx,ry[,phi0,phi1]	
Abrev. :	PE	

	INSTRUCTION GRAPHIQUE	**PRBOX**
Syntaxe :	PRBOX x0,y0,x1,y1	
Abrev. :	PRB	

Explication de cette page :

Les instructions de cette page sont très similaires aux instructions BOX, CIRCLE, ELLIPSE et RBOX. Ici, les figures sont non seulement tracées, mais encore remplies. Le motif de remplissage doit avoir été défini auparavant à l'aide de l'instruction DEFFILL. Vous trouverez plus de précisions dans la description des instructions correspondantes.

FONCTION	PEEK DPEEK LPEEK

Syntaxe :
PEEK(x)
DPEEK(x)
LPEEK(x)

Exemple :

```
A$="A"
D=ARRPTR(A$)
A=LPEEK(D)
L=DPEEK(D+4)
C=PEEK(A)
PRINT "ADRESSE.DES",D
PRINT "ADRESSE",A
PRINT "LONGUEUR",L
PRINT "CODE-ASCII",C
```

Effet : Donne le contenu de 1,2 ou 4 octets de la mémoire principale.

Explication :

'x' est une expression numérique qui représente une adresse de la mémoire principale.

PEEK(x) renvoie le contenu de l'octet situé à l'adresse x.

DPEEK(x) retourne le contenu de deux octets consécutifs dont les adresses respectives sont x et x+1. DPEEK convertit le nombre binaire contenu dans les 16 bits en un nombre décimal.

(Exemple : si le premier octet vaut 11 et le deuxième 189, le résultat obtenu est 11*256+189 = 3005.)

LPEEK(x) est semblable à DPEEK(x), mais se rapporte à 4 octets. ==>

FONCTION	PEEK DPEEK LPEEK suite

(Exemple : le contenu de l'adresse x vaut 8, le contenu de l'adresse x+1 vaut 45, le contenu de l'adresse x+2 vaut 156 et le contenu de l'adresse x+3 vaut 126, LPEEK(x) vaut 8*256^3+45*256^2+156*256+126 = 137206910.)

Dans l'exemple, LPEEK permet d'extraire du descripteur l'adresse à laquelle est mémorisée le contenu de A$. DPEEK retourne la longueur de la chaîne de caractères. Enfin, toujours dans l'exemple, PEEK donne le code ASCII de la lettre 'A'.

(voir à ce sujet l'annexe D :les variables et leur organisation).

FONCTION **PI**

Syntaxe : **PI**

Exemple :
```
INPUT R
U=2*PI*R
F=PI*R*R
PRINT U,F
```

Fonction : Retourne la valeur pi.

Explication :

PI est le rapport du périmètre d'un cercle sur son diamètre. Il vaut environ :

PI=3.1415926536...

L'exemple calcule le périmètre d'un cercle et la surface du disque correspondant, connaissant son rayon.

INSTRUCTION GRAPHIQUE PLOT

Syntaxe : PLOT x,y

Abrev. : PL

Exemple :
```
PLOT 320,200
DEFLINE 1,6,2,2
PLOT 30,30
```

Fonction : Place un point sur l'écran.

Explication :

x et y sont les coordonnées du point, l'origine des coordonnées se trouvant dans le coin supérieur gauche de l'écran. *x* représente la distance horizontale en pixel (0 jusqu'à 639) par rapport au bord gauche de l'écran, *y* la distance verticale (0-399) par rapport au bord supérieur (ces valeurs sont valables en haute résolution). Le premier point de notre exemple est donc affiché au milieu de l'écran.

Cette instruction est identique à l'instruction *DRAW x,y*.

Il faut noter qu'il est parfois nécessaire de redéfinir avec DEFLINE la valeur de la largeur des traits à 1, car sinon aucun point n'est affiché (il y a cependant une exception : si on a choisi avec DEFLINE d'arrondir le début et la fin des lignes, un point sera affiché, dont la taille dépend de la largeur des traits choisie (voir exemple)).

FONCTION POINT

Syntaxe : POINT(x,y)

Exemple :
```
PLOT 100,100
PRINT POINT(99,100)
PRINT POINT(100,100)
```

Fonction : Teste si un point est affiché, ou donne la valeur de sa couleur.

Explication :

(x,y) sont les coordonnées d'un point sur l'écran. L'origine des coordonnées se trouve dans le coin supérieur gauche de l'écran. En haute résolution, le coin inférieur droit a pour coordonnées (639,399).

En haute résolution, la fonction POINT retourne la valeur 1 ou 0, selon que le pixel (point graphique) pointé est 'allumé' ou 'éteint' (car, en haute résolution, il n'existe que les deux valeurs de couleur 0 et 1).

Pour les autres résolutions, la fonction POINT retourne la valeur de la couleur (0 à 15 en basse résolution, et 0 à 3 en moyenne résolution) du point (x,y).pp

INSTRUCTION	POKE DPOKE LPOKE

Syntaxe :
```
POKE x,n
DPOKE x,n
LPOKE x,n
```

Abrev. :
```
PO
DP
LP
```

Exemple :
```
A$="A"
L=ARRPTR(A$)
DPOKE L+4,4
Z=VARPTR(A$)
LPOKE Z,1111638594
PRINT A$
POKE Z,67
PRINT A$
```

Fonction : Ecrit 1, 2 ou 4 octets à l'adresse x de la mémoire principale.

Explication :

'*x*' est une expression numérique qui indique une adresse de la mémoire principale. Pour les instructions DPOKE et LPOKE, x doit être un nombre pair.
'*n*' est une expression numérique dont la valeur est comprise entre 0 et 255 pour POKE, entre 0 et 65535 pour LPOKE et entre -2147483648 et +2147483647 pour LPOKE.

POKE x,n écrit un octet (avec la valeur n) dans la cellule mémoire dont l'adresse est x.
DPOKE x,n charge la valeur n dans les deux cellules mémoires consécutives d'adresses x et x+1. ==>

INSTRUCTION **POKE**
DPOKE
LPOKE
suite

(Exemple : si n vaut 257, la valeur 1 est 'pokée' à la fois à l'adresse x et à l'adresse x+1, car on a : 257 = 1*256 + 1)

LPOKE x,n écrit la valeur n dans les quatre cellules mémoires consécutives dont l'adresse de départ est n. (Exemple : s'il faut poker la valeur 1 dans les quatre cellules mémoires, n devra être égal à 1*256^3 + 1*256^2 + 1*256 + 1, à savoir 16843009).

Comme n ne peut pas dépasser la valeur 2147483647, il faut, pour pouvoir poker un nombre supérieur à 127 dans la première cellule mémoire, travailler avec la complémentation à 2. Exemple : LPOKE x,-1 affecte à toutes les quatre cellules mémoires la valeur 255.

Les instructions décrites ici sont utilisées dans le mode utilisateur de microprocesseur 68000. Cela signifie qu'il est impossible d'écrire dans certaines zones mémoires protégées (en règle générale, les adresses 0 à 2047 et la zone mémoire au delà de 8 Moctets).

Pour pouvoir écrire à ces adresses réservées, il faut appliquer les instructions POKE en mode superviseur : SPOKE, SDPOKE et SLPOKE. (voir instructions correspondantes).

Les instructions inverses de POKE sont les instructions PEEK. Ces dernières travaillent toujours en mode superviseur.

Dans l'exemple, on affecte une nouvelle valeur à la variable A$, à l'aide des instructions POKE (voir à ce sujet l'annexe D).

INSTRUCTION GRAPHIQUE **POLYLINE**
POLYFILL
POLYMARK

Syntaxe :
```
POLYLINE n,X(),Y() [OFFSET X0,Y0]
POLYFILL n,X(),Y() [OFFSET X0,Y0]
POLYMARK n,X(),Y() [OFFSET X0,Y0]
```

Abrev. :
```
POL
POLYF
POLYM
```

Exemple :
```
DIM X(8), Y(8)
FOR I=0 TO 8
  X(I)=320+150*SIN(I*2*PI/8)
  Y(I)=200+150*COS(I*2*PI/8)
NEXT I
DEFFILL 1,2,9
POLYFILL 9,X(),Y()
POLYLINE 9,X(),Y() OFFSET 100,-50
DEFMARK 1,3,30
POLYMARK 9,X(),Y() OFFSET 150,-30
```

Fonction : Trace une ligne brisée passant par n points (en remplissant éventuellement l'intérieur), ou marque les sommets.

Explication :

'*n*' est une expression numérique (dont la valeur maximale est 128). Il indique le nombre de points qui doivent être reliés par des droites. Si l'on veut tracer un polygone fermé, il faut introduire n+1 points (le dernier étant identique au premier), car POLYLINE et POLYFILL sont semblables à DRAW (les instructions POLYLINE et POLYFILL sont cependant sensiblement plus rapides, surtout avec des tableaux d'entiers X%() et Y%()). ==>

INSTRUCTION GRAPHIQUE — **POLYLINE**
POLYFILL
POLYMARK
suite

X( ) et Y( ) sont des vecteurs numériques qui contiennent les coordonnées des n points devant être reliés. Il est important de noter que le premier point est placé dans X(0) et Y(0).

Si on introduit un OFFSET, la totalité de la figure est déplacée de X0 suivant le sens horizontal et de Y0 suivant le sens vertical.

POLYFILL trace une ligne brisée, et en remplit l'intérieur. Le premier et le dernier point sont ici reliés par une droite. Le motif de remplissage doit auparavant être défini avec *DEFFILL*. Cependant, s'il y a des droites qui se coupent, les parties correspondantes ne sont pas remplies. Comme il est difficile de déterminer, dans le cas de figures compliquées dont certaines lignes se recoupent, quelles sont les parties qui doivent être remplies et celles qui doivent rester tel quel, il est conseillé de revoir plus précisément cette partie du programme.

POLYMARK marque les sommets d'une étoile ou d'un autre symbole, que l'on peut définir avec DEFMARK.

FONCTION POS

Syntaxe : POS(n)

Exemple :

```
FOR I=1 TO 600
PRINT "A";
  IF POS(1)=30 THEN
    PRINT
  ENDIF
NEXT I
```

Fonction : Détermine la colonne où se trouve le curseur.

Explication :

'*n*' est un argument muet. Il peut être choisi arbitrairement. La fonction retourne la position horizontale du curseur. La numérotation des colonnes commence ici à 0.

Cependant, la valeur obtenue ne correspond pas toujours avec la position réelle où se trouve le curseur sur l'écran, car seuls les caractères affichés sont comptés et on ne tient pas compte de PRINT AT, des caractères de contrôle, etc.

Seuls CR, LF et BS [CHR$(13, 10, 8)] ont un traitement spécial.

Si une chaîne de caractères dépassant 80 caractères est affichée à l'écran sans utiliser le retour chariot (exemple PRINT SPACE$(100);), POS retournera également une valeur supérieure à 80 (100 dans l'exemple).

INSTRUCTION I/O — **PRINT**
PRINT #

Syntaxe :
PRINT [AT(c,l)][;][expres[,][;]['']]
PRINT #n[,expres[,][;]['']]

Abrev. : P (ou '?')

Exemple :

```
A$="GfA"
B=1986
PRINT A$'
PRINT B,
PRINT A$,B;"GfA"
PRINT
PRINT A$''''B
PRINT At(77,25);A$;
```

Fonction : Permet d'afficher des données à l'écran ou de les envoyer sur le canal n.

Explication :

'expres' est un nombre quelconque d'expressions qui doivent être séparées par des virgules, des points-virgules ou des apostrophes. S'il n'y a pas de séparateur, un point-virgule est automatiquement placé.

Si deux expressions sont séparées par un point-virgule, elles sont affichées dans la même ligne, directement l'une à la suite de l'autre (cela est également valable pour les nombres, car le BASIC GfA ne met aucun caractère blanc ni devant ni derrière les nombres).

Si le séparateur de deux expressions est une virgule, on ajoute à la première expression suffisamment de caractères blancs pour que la longueur totale soit divisible par 16 (c'est-à-dire que l'expression après la virgule commencera à la colonne 17, ou 33, ou 49, etc...).

Si les expressions sont séparées par des apostrophes, des caractères blancs sont affichés en nombre égal (une apostrophe produit le même effet que ;' ';). ==>

INSTRUCTION I/O — **PRINT**
PRINT #
suite

Si la dernière expression n'est suivie par aucun des trois caractères point-virgule, virgule ou apostrophe, un CRLF (retour chariot avec passage à la ligne suivante) est envoyé (c'est-à-dire que le curseur saute au début de la ligne suivante).

Si l'instruction se termine par un point-virgule, une virgule ou une apostrophe, le curseur reste en place.

La partie facultative At(c,l) permet d'afficher des données à un endroit précis de l'écran. L'écran comporte en général 25 lignes (l) et 80 colonnes (c), la numérotation commence en haut à gauche avec (1,1). En donnant la position (c,l) de l'écran, le curseur est positionné à la colonne et à la ligne correspondantes (faites attention quand vous voulez afficher quelque-chose sur la dernière ligne. Pour éviter que l'écran ne se décale d'une ligne (scrolling), il faut mettre un point-virgule à la fin de l'instruction PRINT (voir l'exemple)).

INSTRUCTION I/O PRINT USING

Syntaxe : PRINT USING "format",liste[;]
PRINT #n,USING "format",liste[;]

Abrev. : P USING

Exemple : PRINT AT(7,5);USING "###.##",PI*100
PRINT USING "A!b","lambda"

Fonction : Affiche des nombres et des chaînes de caractères dans un format précis.

Explication :

'*format*' est une expression alphanumérique qui définit le format de l'affichage (voir plus bas).

'*liste*' est une liste d'expressions séparées par des virgules.

Descripteurs de format :

#	réserve une place pour un chiffre
.	position du point décimal
+	affichage des signes 'plus' éventuels
-	réserve une place pour le signe 'moins'
*	les zéros en tête sont remplacés par des *, sinon * est identique à #.
$$	affiche un $ en tête
,	insertion d'une virgule (séparateur des milles)
^^^^	affichage dans le format E+00
^^^^^	format exponentiel E+000
!	le premier caractère d'une chaîne est affiché
&	toute la chaîne est affichée
\..\	affiche un nombre de caractères d'une chaîne égal à la longueur de \..\ (antislashs \ inclus)
_	imprime le caractère suivant

INSTRUCTION DE STRUCTURE PROCEDURE

Syntaxe : PROCEDURE nom [(listevar)]

Abrev. : PRO

Exemple :

```
PRINT "Programme principal"
GOSUB Ssprg(7)
PRINT "retour A=";A
PROCEDURE Ssprg(A)
  PRINT "procédure A=";A
RETURN
PRINT "reste inchangé"
```

Fonction : Indique le début d'une procédure (sous-programme).

Explication :

'nom' est le nom de la procédure. Un nom de procédure peut commencer par un chiffre, à l'inverse des noms de variables. Pour les autres caractères, on peut utiliser : les lettres, les chiffres, le souligné et le point. *'listevar'* sont des noms de variables séparés par des virgules. On transmet à ces variables les paramètres fixés dans l'instruction GOSUB. Les variables de 'listevar' sont toujours des variables locales dans la procédure.

Comme le BASIC GfA n'utilise pas de numéros de lignes, il faut repérer le début d'une procédure (sous-programme). C'est le rôle de cette instruction.

Un autre avantage de cette instruction, en plus des variables locales, est que l'interpréteur peut reconnaître une procédure et éviter ainsi tout traitement du sous-programme non souhaité, car une procédure n'est exécutée que si elle est appelée avec l'instruction GOSUB. ==>

INSTRUCTION DE STRUCTURE PROCEDURE
suite

Si l'interpréteur rencontre l'instruction 'PROCEDURE' dans le déroulement normal du programme, il considère cela comme la fin du programme.

Dans l'exemple, on appelle la procédure dont le nom est 'Ssprg' et on transmet la valeur 7 à la variable locale A. Dans la procédure, on affiche la valeur de A. Pour bien montrer qu'il s'agit d'une variable locale, on procède, après le retour au programme principal, à un nouvel affichage du contenu de A (valeur : 0).

(voir GOSUB, RETURN, LOCAL).

INSTRUCTION PUT

Syntaxe : PUT x0,y0,a$[,mode]

Abrev. : PU

Exemple :

```
FOR I=1 TO 5
CIRCLE 320,200,I*40
NEXT I
GET 0,0,320,399,A$
PUT 320,0,A$,3
```

Fonction : Affiche à l'écran le bloc binaire correspondant à la chaîne de caractères A$ qui a été lue avec GET.

Explication :

(x0,y0) détermine le point de départ, c'est-à-dire le sommet supérieur gauche du bloc à afficher sur l'écran. Ce point correspond au sommet du bloc-écran lu avec l'instruction GET.

'a$' est une variable alphanumérique qui contient le bloc-écran sous la forme d'une suite de bits.

'mode' est une expression numérique d'une valeur comprise entre 0 et 15, qui définit de quelle façon le bloc correspondant à la chaîne a$ doit être affiché sur l'écran :
(C=Chaîne, I=Image précédente)

0: effacer
1: C AND I
2: C AND (NOT I)
3: C (en overwrite)
4: (NOT C) AND I
5: I (inchangé)
6: C XOR I
7: C OR I
8: NOT (C OR I)
9: NOT (C XOR B)
10: NOT I (inverse-vidéo)
11: C OR (NOT I)
12: NOT C (overwrite en inv.vid.)
13: (NOT C) OR I
14: NOT (C AND I)
15: 1

INSTRUCTION I/O PUT

Syntaxe : PUT [#]n[,i]

Abrev. : PU

Exemple : voir FIELD

Fonction : Ecrit un enregistrement sur un fichier à accès direct.

Explication :

'*n*' est une expression entière entre 0 et 99 qui indique le numéro d'un canal de données préalablement ouvert avec l'instruction OPEN.

Le paramètre facultatif i est une expression entière comprise entre 1 et le nombre d'enregistrements du fichier plus un (max. 65535) et correspond au numéro de l'enregistrement dans le fichier. Si 'i' est absent, l'enregistrement sera placé en fin de fichier.

INSTRUCTION QUIT

Syntaxe : QUIT

Abrev. : Q

Exemple :
```
PRINT "Entrez le mot de passe !"
INPUT E$
IF E$<>"GfA" THEN
  QUIT
ENDIF
```

Fonction : Permet de quitter l'interpréteur.

Explication :

Si l'interpréteur rencontre l'instruction *QUIT* au cours de l'exécution d'un programme, il rend la main au desktop du GEM.

Cette instruction est identique à l'instruction *SYSTEM*.

FONCTION **RANDOM**

Syntaxe : RANDOM(x)

Exemple :

```
REPEAT
  L=RANDOM(6)+1
  PRINT L
UNTIL L=3
```

Fonction : Retourne un nombre entier aléatoire compris entre 0 (inclus) et x.

Explication :

'*x*' est un nombre quelconque (même négatif). *RANDOM(x)* donne un nombre entier aléatoire compris entre 0 (inclus) et x (exclus). Pour garantir une répartition de Laplace (équiprobabilité de tous les évènements), il faut donner à x une valeur entière.

RANDOM(x)=TRUNC(x*RND)

L'exemple simule les lancés successifs d'un dé jusqu'à ce qu'on obtienne un 3.

INSTRUCTION GRAPHIQUE RBOX

Syntaxe : RBOX x0,y0,x1,y1

Abrev. : RB

Exemple : RBOX 50,50,590,350

Fonction : Trace un 'rectangle' dont les coins sont arrondis et dont les 'sommets' diagonalement opposés ont pour coordonnées (x0,y0) et (x1,y1).

Explication :

Pour repérer les 'sommets', on prend pour origine des coordonnées le coin supérieur gauche. Les sommets ne doivent pas nécessairement se trouver à l'intérieur de l'écran. Dans ce cas, seule une partie du 'rectangle' est représentée.

(voir aussi DEFLINE).

INSTRUCTION **READ**

Syntaxe : READ var[,var]...

Abrev. : REA

Exemple :
```
READ P,N$,Z%
PRINT P'N$'Z%
DATA 75006,PARIS,6
```

Fonction : Lit les valeurs contenues dans une instruction DATA et les affectent aux variables 'var'.

Explication :

'var' sont des variables numériques, booléennes ou alphanumériques quelconques. Cette instruction ne peut être utilisée qu'en relation avec l'instruction DATA.

Lorsque l'interpréteur rencontre dans un programme l'instruction READ pour la première fois, il lit la première valeur de la première ligne-DATA et l'affecte à la (ou à la première) variable de l'instruction READ. Pour les instructions READ suivantes, on traite les données de la (des) ligne(s)-DATA dans l'ordre où elles ont été introduites.

Deux points sont à noter :

- Les données contenues dans les lignes-DATA doivent être du même type que les variables qui vont contenir ces valeurs.

- Le nombre des données dans les lignes-DATA doit être égal (ou supérieur) au nombre des variables 'var' qui suivent les instructions READ (si ce nombre est supérieur, les données supplémentaires ne sont naturellement pas prises en compte).

(voir absolument DATA et RESTORE).

INSTRUCTION I/O — RELSEEK

Syntaxe : RELSEEK [#]n,x

Abrev. : REL

Exemple :
```
OPEN "O",#1,"DAT"
PRINT #1,"1234567890"
SEEK #1,8
RELSEEK #1,-5
PRINT LOC(#1)
```

Fonction : Déplace le pointeur du fichier, dont le numéro de canal est n, de x octets.

Explication :

'n' est une expression entière comprise entre 0 et 99 qui correspond au numéro d'un canal de données préalablement ouvert avec OPEN.

'x' est une expression entière dont la valeur est telle que : 0 <= LOC(#n) + x <= LOF(#n).

A chaque canal de données, correspond un pointeur de fichier (pointeur d'écriture et de lecture) qui est positionné sur un octet précis dans le fichier. Avec RELSEEK, on peut déplacer ce pointeur relativement à sa position précédente. Si x est positif, on déplace le pointeur de fichier de x octets vers la fin du fichier. Si x est négatif, le déplacement se fait dans le sens opposé, c'est-à-dire vers le début du fichier.

INSTRUCTION REM

Syntaxe : REM texte

Abrev. : R (ou: ')

Exemple :
```
REM Calcul de l'énergie cinétique
LET E=(M*V*V)/2
```

Fonction : Permet d'insérer des commentaires à l'intérieur du programme.

Explication :

Les commentaires illustrent un programme et rendent sa compréhension plus facile.

L'instruction *REM* n'est pas exécutée, mais le texte *'texte'* apparait sur le listing.

Un commentaire peut être placé après une instruction, en utilisant le caractère séparateur "!" :

```
DO      !Boucle sans fin
LOOP    !Avec rien dedans
```

Ces commentaires ne sont bien sûr pas admis après DATA (et REM).

INSTRUCTION DE STRUCTURE REPEAT...UNTIL

Syntaxe :
```
REPEAT
UNTIL condition
```

Abrev. :
```
REP
U
```

Exemple :
```
REPEAT
  A=A+1
  PRINT A
UNTIL A=20
```

Fonction : Réalise une boucle conditionnelle.

Explication :

La partie du programme qui se trouve entre REPEAT et UNTIL est exécutée plusieurs fois jusqu'à ce que la condition soit remplie. A l'inverse de l'instruction WHILE...WEND, la condition n'est testée qu'en fin de boucle. C'est pourquoi une boucle REPEAT...UNTIL est toujours parcourue au moins une fois.

INSTRUCTION RESERVE

Syntaxe : RESERVE n

Abrev. : RESE

Exemple : (voir EXEC).

Fonction : Diminution ou augmentation de la place mémoire allouée au BASIC GfA.

Explication :

'n' est une expression numérique qui détermine la nouvelle valeur de FRE(0).

Comme HIMEM est un multiple de 256 octets, FRE(0) sera peut-être un peu plus grand (de 255 octets au maximum). Cette instruction permet d'éviter que le BASIC n'empiète sur des zones mémoires que l'on veut se réserver pour des programmes en langage machine ou des fichiers RSC. Cependant, pour réserver des zones mémoires contenant uniquement des données, il est préférable d'utiliser l'instruction MALLOC() (adr=GEMDOS(&48,L:nombre) pour éviter que ces zones ne soient utilisées par le GEM (sélections de fichiers, alarmes ...).

(voir aussi HIMEM, EXEC).

INSTRUCTION **RESTORE**

Syntaxe : RESTORE [étiquette]

Abrev. : RES

Exemple :

```
READ A,B,C,D,E
RESTORE
READ F,G,H,I
RESTORE Etiq
READ J,K,L,M
PRINT A'B'C'D'E"F'G'H'I"J'K'L'M
DATA 1,2,3
Etiq:
DATA 4,5,6,7
```

Fonction : Positionne le pointeur-DATA au début du programme ou après 'étiquette:'.

Explication :

'étiquette' est une suite de caractères composée de chiffres, de lettres alphabétiques, du souligné et du point. Le premier caractère peut être un chiffre, à l'inverse des noms de variables.

L'instruction *RESTORE* permet de relire une ou plusieurs lignes-DATA. Le pointeur-DATA, qui pointe sur la prochaine donnée devant être lue, est repositionné au tout début (avec RESTORE) ou au début de la ligne-DATA située après 'étiquette:' (avec RESTORE 'étiquette'). (Il est important de noter que l'identificateur servant de marque entre les lignes-DATA est constitué du nom de l'étiquette suivi d'un double point).

(voir aussi READ et DATA).

INSTRUCTION RESUME

Syntaxe : RESUME
RESUME NEXT
RESUME étiquette

Abrev. : RESU

Exemple : voir ON ERROR GOSUB.

Fonction : Retour d'une procédure d'erreur.

Explication :

RESUME exécute à nouveau l'instruction ayant conduit à l'erreur.

RESUME NEXT relance l'exécution du programme après l'instruction ayant entrainé l'erreur.

'RESUME étiquette' branche le programme après *'étiquette:'*. Si l'étiquette (on dit aussi label) se trouve dans le programme principal, la pile des 'RETURNs' est effacée et les variables globales sont restaurées.

Pour des erreurs FATALes, seule l'instruction *'RESUME étiquette'* est possible.

INSTRUCTION DE STRUCTURE RETURN

Syntaxe : RETURN

Abrev. : RET

Exemple :
```
PRINT "Programme principal"
GOSUB Ssprg
PRINT "De retour"
PROCEDURE Ssprg
  PRINT "Procédure"
RETURN
```

Fonction : Indique la fin d'un sous-programme (procédure).

Explication :

Lorsque l'interpréteur rencontre l'instruction *RETURN* à l'intérieur d'une procédure (sous-programme), le traitement de la procédure est terminé et le programme se branche après l'instruction *GOSUB* ayant appelé la procédure.

Les variables locales sont alors effacées.

(voir GOSUB, PROCEDURE, LOCAL).

FONCTION **RIGHT$**

Syntaxe : RIGHT$(string[,n])

Exemple :
```
N$="BASIC GfA"
PRINT RIGHT$(N$)
PRINT RIGHT$(N$,5)
PRINT RIGHT$(N$,12)
```

Fonction : Extrait le dernier ou les n derniers caractères (de droite) de la chaîne 'string'.

Explication :

'*string*' est une chaîne de caractères ou une variable de ce type.

'*n*' est normalement un nombre entier ou une variable entière (cependant, il est possible de donner à n une valeur décimale. Seule sa partie entière sera prise en compte par l'éditeur).

Si le paramètre facultatif n est absent, la fonction retourne le dernier caractère de la chaîne 'string'.

Sinon, la fonction extrait la sous-chaîne composée des n derniers caractères (de droite) de 'string' (les caractères blancs sont pris en compte). Si n est supérieur au nombre total de caractères formant la chaîne 'string', on obtient la chaîne 'string' elle-même. Si n=0, on obtient la chaîne vide.

INSTRUCTION I/O **RMDIR**

Syntaxe : RMDIR "nomrep"

Abrev. : RM

Exemple :
```
MKDIR "A:REP"
FILES "A:*.*"
RMDIR "A:REP"
FILES "A:*.*"
```

Fonction : Détruit le répertoire 'nomrep', à condition qu'il soit vide.

Explication :

'nomrep' est le nom du répertoire qu'on veut détruire. Ce nom peut contenir toutes les spécifications relatives à l'organisation hiérarchique des fichiers (\):

Si 'nomrep' est simplement un nom de répertoire (non précédé du caractère \), il doit appartenir au répertoire (principal ou non) où l'on se trouve (pour changer de répertoire, utiliser CHDIR).

Si l'on veut détruire un répertoire qui appartient à un répertoire précis, il faut préciser le chemin. Ce chemin commence toujours par le caractère \, si l'on part de la directory (répertoire) principale. Ainsi par exemple, l'instruction RMDIR "\KFZ\PARTNR" a pour effet de détruire le répertoire PARTNR appartenant au répertoire KFZ de la directory principale (on ne peut évidemment détruire un répertoire que s'il existe (voir EXIST) et s'il ne contient aucun fichier).

Si l'on choisit une unité de disquette différente de l'unité prise par défaut, il faut faire précéder le nom du répertoire par la lettre correspondant au lecteur, suivi d'un double-point (ex : 'B:REP').

FONCTION **RND**

Syntaxe : RND[(x)]

Exemple :
```
FOR I=1 TO 20
  PRINT RND
NEXT I
```

Fonction : Retourne un nombre aléatoire entre 0 et 1.

Explication :

Le paramètre facultatif (x) n'est pas pris en compte. RND retourne un nombre aléatoire entre 0 (inclus) et 1 (exclus).

En utjlisant la formule :

Z=INT(RND*n)+1 ,

on obtient un nombre entier aléatoire compris entre 1 et n (tous les deux inclus).

(voir RANDOM).

INSTRUCTION **RSET**

Syntaxe : RSET var=string

Abrev. : RS

Exemple :
```
A$="AAAAAAAAA"
B$=SPACE$(7)
C$="GfA"
RSET A$=C$
RSET B$="BASIC GfA"
PRINT A$
PRINT B$
```

Fonction : Place la chaîne de caractères 'string' dans la partie droite de la chaîne 'var'.

Explication :

'*var*' est une variable alphanumérique.

'*string*' est une expression alphanumérique quelconque.

La chaîne de caractères 'string' est placée dans la chaîne 'var', l'opération s'effectuant par la droite. Si la longueur de 'string' est inférieur à celle de 'var', la partie de 'var' qui n'a pas été touchée est remplie de caractères blancs. Dans le cas contraire, 'string' est tronqué à droite.

RSET est normalement utilisé en liaison avec l'instruction FIELD pour la création d'un fichier à accès direct. Dans ce cas, les données numériques doivent être converties en chaînes de caractères avec MKI$, MKL$, MKS$, MKF$ ou MKD$, avant de pouvoir leurs appliquer l'instruction RSET.

(voir aussi LSET).

INSTRUCTION **RUN**

Syntaxe : RUN

Abrev. : RU

Exemple :

```
PRINT "Appuyez sur une touche !"
IF INPUT$(1)="s" THEN
  END
ELSE
  RUN
ENDIF
```

Fonction : Lance l'exécution du programme.

Explication :

L'instruction RUN exécute le programme à partir de la première ligne de programme.

INSTRUCTION I/O **SAVE**
PSAVE

Syntaxe : SAVE "nomfich"
PSAVE "nomfich"

Abrev. : SA
PS

Exemple : SAVE "A:\PROG"
FILES "A:*.*"

Fonction : Sauvegarde un programme sur disquette (avec autostart pour PSAVE).

Explication :

'*nomfich*' est le nom du programme.

Si on veut sauvegarder le programme sur une unité de disquette déterminée, par exemple l'unité A, on utilisera le préfixe 'A:'. 'nomfich' peut également contenir toutes les spécifications relatives à l'organisation hiérarchique des fichiers.
Exemple : l'instruction SAVE "B:\PROGR\TEST" signifie que le programme TEST.BAS va être sauvegardé sur la disquette de l'unité B dans le répertoire PROGR. (si aucune extension n'est donnée, le BASIC GfA ajoute automatiquement le suffixe .BAS).

Lors du chargement en mémoire principale, les programmes qui ont été sauvegardés avec PSAVE ne sont pas listés mais sont automatiquement lancés.

Attention ! si la disquette contient un programme de même nom, l'ancienne version est effacée avant de sauvegarder la nouvelle.

Si les guillemets qui entourent 'nomfich' manquent, ils sont automatiquement mis en place.

INSTRUCTION I/O SEEK

Syntaxe : SEEK [#]n,i

Abrev. : SEE

Exemple :

```
OPEN "O",#1,"DAT1"
OPEN "O",#2,"DAT2"
PRINT #1,"1234567"
PRINT #2,"ABCDEF"
SEEK #1,3
SEEK #2,5
PRINT LOC(#1),LOC(#2)
```

Fonction : Positionne le pointeur de fichier sur le ième octet du fichier correspondant au canal n.

Explication :

'*n*' est une expression entière comprise entre 0 et 99 qui se rapporte au numéro d'un canal de données ouvert préalablement avec OPEN.

'*i*' est une expression entière dont la valeur doit être inférieure ou égale à la taille (en nombre d'octets) du fichier.

A chaque canal de données est associé un pointeur de fichier (pointeur de lecture et d'écriture) qui est positionné sur un octet précis du fichier. Avec SEEK, on peut placer ce pointeur sur n'importe quel octet du fichier. Plus précisément, si i est positif, on place le pointeur sur le ième octet en partant du début du fichier. Si i est négatif, on compte les octets en partant de la fin du fichier.

INSTRUCTION GRAPHIQUE SETCOLOR

Syntaxe : SETCOLOR i,r,v,b ou
SETCOLOR i,n

Abrev. : SE

Exemple :
```
SETCOLOR 0,0,0,0
FOR I=1 TO 30000
NEXT I
SETCOLOR 0,1911
```

Fonction : Fixe les parts du rouge, du vert et du bleu dans le registre de couleur numéro i.

Explication :

Suivant la résolution graphique choisie, on dispose d'un nombre, plus ou moins grand, de registres de couleur (registres 0 à 15 en basse résolution, registres 0 à 3 en moyenne résolution, registres 0 et 1 en haute résolution).

'i' est le numéro du registre, *'r','v' et 'b'* sont les parts des couleurs rouge, verte et bleue (toujours comprises entre 0 et 7).

Une autre possibilité pour définir une couleur est d'utiliser la valeur *'n'* calculée de la façon suivante : n = r*256 + v*16 + b.

En haute résolution, les seules instructions efficaces sont *SETCOLOR 0,0 et SETCOLOR 0,1* (si malgré tout on donne une autre valeur à n, les valeurs paires de n auront le même effet que n=0 et les valeurs impaires le même effet que n=1).

INSTRUCTION SETTIME

Syntaxe : SETTIME timestring,datestring

Abrev. : SETT

Exemple :
```
PRINT TIME$,DATE$
SETTIME "15:30:11","29.6.1986"
PRINT TIME$,DATE$
```

Effet : Met en place l'heure et la date.

Explication :

'timestring' est une expression alphanumérique qui contient l'heure.

'timestring' est composé des valeurs 'heures, minutes et secondes' séparées par des double-points. Comme chacun des trois termes nécessite deux caractères, les double-points peuvent disparaître. La donnée des secondes est également facultative, de telle sorte que "1030", par exemple, est identique à "10:30:00".

'datestring' est une expression alphanumérique indiquant la date. Elle doit toujours contenir : le jour, le mois et l'année séparés entre eux par des points. Pour les années entre 1980 et 2079, les deux premiers chiffres sont facultatifs.

Si l'heure et la date sont données sous un format incorrect, elles ne sont pas modifiées par la suite.

INSTRUCTION GRAPHIQUE SGET

Syntaxe : SGET var

Abrev. : SG

Exemple :

```
SGET SCREEN1$
FOR I=1 TO 100
  PRINT I'
NEXT I
SGET SCREEN2$
DO
  SPUT SCREEN1$
  SPUT SCREEN2$
LOOP
```

Fonction : Lecture rapide de la mémoire-écran dans une chaîne de caractères.

Explication :

'var' est une variable alphanumérique (variable chaîne de caractères). Avec SGET, la totalité de la mémoire-écran (32000 octets) est copiée dans 'var' (avec un BMOVE). Cette instruction est encore un peu plus rapide que GET 0,0,xm,ym,A$ et ne dépend pas de la résolution graphique choisie. Dans l'exemple, on copie le contenu de la mémoire-écran dans une variable alphanumérique, on affiche ensuite quelques nombres pour réaliser une nouvelle sauvegarde de la mémoire-écran dans une deuxième variable, et pour finir on affiche alternativement l'une et l'autre variables, ce qui a pour effet de faire scintiller l'écran.

(voir aussi SPUT, GET, PUT et BMOVE).

FONCTION SGN

Syntaxe : SGN(x)

Exemple :
```
INPUT A
B=SGN(A)*A
PRINT B
```

Fonction : Teste si x est positif, négatif ou nul.

Explication :

'*x*' est une expression numérique quelconque.

SGN(x) est la fonction mathématique Signe.

Elle retourne, selon le cas, la valeur -1, 0 ou 1, à savoir :

```
 0    , si x=0
+1    , si x>0
-1    , si x<0
```

L'exemple simule la fonction mathématique valeur absolue (voir ABS).

INSTRUCTION GRAPHIQUE SHOWM

Syntaxe : SHOWM

Abrev. : SH

Exemple :
```
SHOWM
DO
  PLOT MOUSEX,MOUSEY
LOOP
```

Fonction : Mise en place du curseur de la souris.

Explication :

Après une hardcopy ou un HIDEM, le curseur de la souris disparait de l'écran.

Cette instruction permet de faire réapparaître la souris (après une hardcopy, la souris ne redevient visible qu'après l'avoir déplacer). Aprés un appel au VDI ou d'autres opérations, la souris disparait à nouveau et redevient visible après SHOWM ou aprés l'avoir déplacé. L'exemple permet de dessiner avec une souris visible.

(voir HIDEM).

FONCTION SIN

Syntaxe : SIN(x)

Exemple :

1)
```
INPUT RAD
PRINT SIN(RAD)
INPUT DEGRE
PRINT SIN(DEGRE*PI/180)
```

2)
```
PLOT 320,200
FOR I=1 TO 5400
  X=I/36*COS(I*PI/180)
  Y=I/36*SIN(I*PI/180)
  DRAW TO 320+X,200+Y
NEXT I
```

Fonction : Retourne la valeur du sinus de x.

Explication :

'*x*' est une expression numérique qui indique l'angle (en radians) dont on veut calculer le sinus.

Si l'on veut introduire un angle en degrés, on remplace 'x' par 'deg*PI/180', où PI est une fonction du BASIC GfA qui donne la valeur de la constante universelle pi.

Le premier exemple calcule le sinus d'un angle donné en radians, puis le sinus d'un angle donné en degrés.

Le deuxième exemple dessine une spirale degré par degré.

INSTRUCTION SOUND

Syntaxe : SOUND canal,volume,note,octave[,durée]
SOUND canal,volume,#periode[,durée]

Abrev. : SO

Exemple :

```
FOR I=1 TO 8
  SOUND 1,15,1,1,40
  FOR J=1 TO 6
    READ A
    SOUND 1,15,A,1,20
  NEXT J
  RESTORE
NEXT I
DATA 3,5,6,8,10,12
```

Effet : Génère des notes de musique.

Explication :

'canal' est une expression qui peut prendre les trois valeurs 1, 2 et 3 et qui détermine lequel des trois canaux-sons possibles va être utilisé.

'volume' est une expression numérique dont la valeur est comprise entre 0 et 15, et indique le volume sonore.

'note' est une expression pouvant valoir 1 à 12, et correspond à la note de musique :

1 = DO	7 = FA #
2 = DO #	8 = SOL
3 = RE	9 = SOL #
4 = MI b	10 = LA
5 = MI	11 = SI b
6 = FA	12 = SI

INSTRUCTION **SOUND**
suite

'octave' est une expression dont la valeur est comprise entre 1 et 8, et détermine l'octave dans laquelle la note **'note'** est jouée. Le LA du diapason (440 Hz) se trouve dans la quatrième octave.

'durée' est une expression entière, et indique le temps en 1/50 secondes pendant lequel le BASIC GfA interrompt le programme avant d'exécuter l'instruction suivante.

Une autre possibilité est offerte pour définir la hauteur du son : au lieu de donner la note et l'octave, on peut entrer la période 'période' précédée du caractère '#'. La période est calculée à partir de la fréquence comme suit :

```
Periode = TRUNC(125000/Frequence + 0.5)
```

Le LA du diapason (440 Hz), par exemple, correspond à une période valant 284. Ce son, joué pendant 10 secondes, peut donc être généré de deux manières différentes :

```
SOUND 1,15,10,4,500
SOUND 1,15,#284,500
```

Dans l'exemple, un morceau de musique est joué sur les 8 octaves.

(voir aussi WAVE).

FONCTION **SPACE$**

Syntaxe : SPACE$(x)

Exemple :
```
FOR I=0 TO 20
  PRINT SPACE$(I),I
NEXT I
```

Fonction : Définit une chaîne de caractères composée de x caractères blancs.

Explication :

'*x*' est une expression numérique dont la valeur doit être comprise entre 0 et 32767. Seul est pris en compte la partie entière de x.

Cette instruction donne le même résultat que STRING$(x," ").

FONCTION **SPC**

Syntaxe : SPC(n)

Exemple :
```
PRINT "BASIC";SPC(10);"GfA"
PRINT "ici";SPC(20);"là-bas"
```

Fonction : Utilisée dans une instruction PRINT, cette fonction affiche n caractères blancs.

Explication :

'*n*' est une expression entière. On ne prend en compte que la valeur TRUNC(n MOD 256).

La fonction SPC ne peut être appliquée qu'en liaison avec l'instruction PRINT.

INSTRUCTION **SPOKE**
SDPOKE
SLPOKE

Syntaxe :
SPOKE x,n
SDPOKE x,n
SLPOKE x,n

Abrev. :
SP
SD
SL

Exemple :
```
A$="A"
L=ARRPTR(A$)
SDPOKE L+4,4
Z=VARPTR(A$)
SLPOKE Z,1111638594
PRINT A$
SPOKE Z,67
PRINT A$
```

Fonction : Ecrit 1, 2 ou 4 octets dans une zone mémoire dont l'adresse de début est x.

Explication :

'*x*' est une expression numérique qui indique une adresse de la mémoire principale. Pour les instructions SDPOKE et SLPOKE, x doit être un nombre pair.
'*n*' est une expression numérique dont la valeur varie entre 0 et 255 pour SPOKE, entre 0 et 65535 pour SDPOKE, et entre -2147483648 et +2147483647 pour SLPOKE.

Ces instructions travaillent dans le mode superviseur du microprocesseur 68000, c'est-à-dire qu'on peut aussi accéder aux zone-mémoires protégées. Pour le reste, ces instructions ont les mêmes fonctions que les instructions POKE qui travaillent en mode utilisateur (voir description correspondante).

INSTRUCTION GRAPHIQUE SPRITE

Syntaxe : SPRITE A$[,x,y]

Abrev. : SPR

Exemple :

```
AS=MKI$(1)+MKIS(1)+MKIS(0)
A$=A$+MKI$(0)+MKI$(1)
FOR I=1 TO 16
  A$=A$+MKI$(0)+MKIS(65535)
NEXT I
SPRITE A$,20,20
PAUSE 200
SPRITE A$
```

Fonction : Affiche un sprite défini dans A$ à la position (x,y), ou bien l'efface.

Explication :

'A$' est une chaîne de définition (voir DEFMOUSE). Pour la première utilisation de SPRITE, cette chaîne est composée de 74 caractères (2*(5+2*16)), de la manière suivante :

```
A$ = MKI$(coord. x du point d'action)
   + MKI$(coord. y du point d'action)
   + MKI$(0) normal  ou  MKIS(1) XOR
   + MKI$(couleur du masque)  souvent 0
   + MKIS(couleur du sprite)  souvent 1
   + B$  (motifs binaires du masque et du sprite)
```

A l'inverse de DEFMOUSE, les motifs binaires du masque et du sprite ne sont pas mémorisés dans B$ consécutivement, mais alternativement (voir absolument DEFMOUSE).

'x' et 'y' sont des expressions numériques qui indiquent les coordonnées du point d'action du sprite. Si le même sprite est affiché une deuxième fois à un autre endroit, le premier sprite est effacé. Si les paramètres facultatifs x et y manquent, le sprite disparaît totalement de l'écran.

INSTRUCTION GRAPHIQUE SPUT

Syntaxe : SPUT var

Abrev. : SPU

Exemple : (voir SGET).

Fonction : Copie rapide d'une chaîne de 32000 caractères dans la mémoire-écran.

Explication :

'var' est une variable alphanumérique. SPUT copie (à l'aide de BMOVE) la zone mémoire de 32000 octets, qui commence à l'adresse de de la variable 'var', dans la mémoire-écran.

(voir aussi SGET, PUT, GET et BMOVE).

FONCTION **SQR**

Syntaxe : SQR(x)

Exemple :
```
INPUT X
INPUT Y
DISTANCE = SQR(X*X+Y*Y)
PRINT DISTANCE
```

Fonction : Calcule la racine carrée de x.

Explication :

'*x*' est une expression numérique dont la valeur doit être supérieure ou égale à 0.

SQR(x) calcule la racine carrée de x.

Dans l'exemple ci-dessus, on calcule la distance d'un point de coordonnées (X,Y) par rapport au point origine.

Si on veut calculer la racine n-ième d'un nombre avec n<>2 (par ex. la racine troisième), il faut se servir de la formule mathématique suivante :

Racine n-ième de x = x^(1/n)

Ainsi, la racine troisième de 8 est égale à 8^(1/3) = 2.

INSTRUCTION **STOP**

Syntaxe : STOP

Abrev. : ST

Exemple :
```
OPEN "O",#1,"DAT"
STOP
(Après exécution du programme, entrer en
mode direct :)
PRINT #1,"BASIC"
```

Fonction : Arrête l'exécution du programme.

Explication :

L'instruction STOP peut être utilisée n'importe où dans le programme. La différence avec l'instruction END est double. Tout d'abord, STOP ne ferme pas les fichiers (voir exemple). Ensuite, on peut reprendre l'exécution du programme là où il a été interrompu, grâce à l'instruction CONT.

FONCTION **STR$**

Syntaxe : STR$(x)

Exemple :
```
A=3.5E+56
B=&O22
PRINT STR$(A)
PRINT STR$(234)
PRINT STR$(B)
```

Fonction : Convertit la valeur numérique x en une chaîne de caractères.

Explication :

'x' est un nombre dans une représentation quelconque (c'est-à-dire l'écriture décimale normale (sans préfixe), l'écriture hexadécimale (préfixe & ou &H), la représentation octale (préfixe &O), la représentation binaire (préfixe &X)) ou une variable numérique.

STR$(x) convertit la valeur x en une chaîne de caractères correspondant à l'écriture décimale de x (c'est-à-dire que la variable B de l'exemple devient la chaîne "18").

(voir aussi HEX$(x), OCT$(x) et BIN$(x)).

FONCTION **STRING$**

Syntaxe : STRING$(n,string) ou
STRING$(n,c)

Exemple :
```
Z$="A"
PRINT STRING$(50,Z$)
PRINT STRING$(50,"A")
PRINT STRING$(50,65)
PRINT STRING$(25,"AA")
```

Effet : Fournit une chaîne de caractères qui est constituée de n fois la chaîne 'string' ou de n fois CHR$(c).

Explication :

'*n*' est un nombre entre 0 et 32767.

'*string*' est une expression alphanumérique quelconque.

'*c*' est une expression numérique qui représente le code ASCII d'un caractère.

Une table de correspondance entre les caractères et leurs codes est donnée en annexe C.

'c' est transformé par l'interpréteur en un nombre entier compris entre 0 et 255, d'après la formule c MOD 256 pour les nombres positifs et la formule (c+2147483648) MOD 256 pour les nombres négatifs.

La fonction retourne une chaîne de caractères qui est constituée de n fois de la chaîne 'string' ou du caractère CHR$(c).

La longueur de la chaîne obtenue ne doit pas dépasser la valeur 32767. L'exemple donne dans tous les quatre cas une chaîne de caractères composée de 50 fois la lettre 'A'.

INSTRUCTION ARITHMETIQUE SUB

Syntaxe : SUB var,n

Abrev. : S

Exemple :

```
T=TIMER
FOR I%=1 TO 10000
  SUB A%,5
NEXT I%
PRINT (TIMER-T)/200
A%=0
T=TIMER
FOR I%=1 TO 10000
  A%=A%-5
NEXT I%
PRINT (TIMER-T)/200
```

Fonction : Soustrait n à la valeur de 'var'.

Explication :

'*var*' doit être une variable numérique ou un élément d'un tableau numérique.

'*n*' est une expression numérique.

SUB var,n est identique à var=var-n. L'avantage de l'instruction SUB réside dans la vitesse d'exécution (testez l'exemple !). Ainsi, l'utilisation de l'instruction SUB augmente de façon sensible la vitesse d'exécution déjà très élevée par ailleurs (presque du simple au double dans l'exemple).

INSTRUCTION **SWAP**

Syntaxe : SWAP var1,var2

Abrev. : SW

Exemple :
```
DIM A(3),B(8)
ARRAYFILL B(),8
A$="premier"
B$="deuxième"
SWAP A$,B$
SWAP A(),B()
PRINT A$,B$
PRINT A(8),B(2)
```

Fonction : Echange les contenus de 'var1' et de 'var2'.

Explication :

Cet échange peut s'effectuer avec des variables de n'importe quel type : *var1 et var2* peuvent aussi bien être des variables et tableaux numériques que des variables et tableaux alphanumériques ou encore des variables et tableaux booléens.

'var1' et 'var2' doivent appartenir au même type.

S'il s'agit de tableaux, il y a aussi échange des dimensions (voir exemple).

INSTRUCTION **SYSTEM**

Syntaxe : SYSTEM

Abrev. : SY

Exemple :
```
PRINT "Entrez le mot de passe !"
INPUT E$
IF E$<>"GfA" THEN
  SYSTEM
ENDIF
```

Fonction : Permet de quitter l'interpréteur.

Explication :

Si, lors du déroulement du programme, l'interpréteur rencontre l'instruction *SYSTEM*, il rend la main au Desktop du GEM.

Cette instruction est identique à *QUIT*.

FONCTION **TAB**

Syntaxe : TAB(n)

Exemple :
```
PRINT TAB(30);"GfA"
PRINT SPACE$(40)
PRINT TAB(30);"BASIC"
```

Fonction : Place une tabulation à la n-ième colonne.

Explication :

'n' est une expression entière. Cependant, on ne prend en compte que la valeur TRUNC(n MOD 256).

La fonction TAB ne peut être appliquée qu'en liaison avec l'instruction PRINT.

Si le curseur se trouve déjà à la colonne n, TAB a pour effet de placer le curseur à la n-ième colonne de la ligne suivante.

FONCTION	TAN

Syntaxe : TAN(x)

Exemple :
```
INPUT RAD
PRINT TAN(RAD)
INPUT DEGRE
PRINT TAN(DEGRE*PI/180)
```

Fonction : Calcule la tangente de x.

Explication :

'x' est une expression numérique qui représente l'angle (en radians) dont on veut calculer la tangente.

Si l'on veut entrer l'angle en degrés, il faut remplacer x par 'deg*PI/180', où PI est une fonction du BASIC GfA qui retourne une valeur approchée de la constante universelle pi.

Dans l'exemple, on calcule la tangente d'un angle donné en radians, puis celle d'un angle donné en degrés.

INSTRUCTION **TEXT**

Syntaxe : TEXT x,y,[l,]string

Abrev. : T

Exemple :
```
DEFTEXT 1,16,0,21
A$="ABC D"
TEXT 10,50,A$
TEXT 10,350,200,A$
TEXT 300,350,-200,"ABC D"
```

Fonction : Affiche un texte, en mode graphique, à la position (x,y) de l'écran.

Explication :

Les caractéristiques du texte graphique peuvent être définies avec l'instruction DEFTEXT.

Le texte est affiché à partir de la position (x,y). L'origine des coordonnées se trouve dans le coin supérieur gauche de l'écran. Le point (x,y) lui-même correspond au coin inférieur gauche de la portion de texte.

Le paramètre facultatif l peut être négatif ou positif, et détermine la longueur de la zone-écran qui va contenir le texte. Si l est positif, on donne au texte une longueur l, en faisant varier les espaces entre les caractères ; si l est négatif, on fait varier les espaces entre les mots (voir exemple). Si l=0, le texte est affiché normalement.

'*string*' représente une chaîne de caractères ou une variable alphanumérique (voir exemple).

FONCTION **TIME$**

Syntaxe : TIME$

Exemple :

```
DEFTEXT 1,16,0,32
DO
  PRINT AT(1,1);TIME$
  TEXT 240,180,TIME$
LOOP
```

Fonction : Permet d'obtenir l'heure interne du système.

Explication :

La fonction retourne une chaîne de caractères indiquant l'heure interne du système qui est contenue dans le tableau de contrôle. Cette chaîne a le format suivant :

hh:mm:ss

Les secondes augmentent par pas de 2.

FONCTION **TIMER**

Syntaxe : TIMER

Exemple :

```
T=TIMER
REPEAT
  Z=INT((TIMER-T)/2)/100
  PRINT AT(38,3);Z"
UNTIL INKEY$<>""
```

Fonction : Calcule et transmet le temps écoulé depuis la mise en route du système, en 1/200 de secondes.

Explication :

L'appel de cette fonction permet d'obtenir le temps écoulé depuis la mise en fonctionnement du système. Cette durée varie par pas de 1/200 de secondes.

L'exemple simule un chronomètre affichant les centièmes de secondes.

INSTRUCTION **TITLEW**

Syntaxe : TITLEW n,"titre"

Abrev. : TIT

Exemple : TITLEW 2,"Nouveau TITRE"

Fonction : Permet de donner un (nouveau) titre à la fenêtre numéro n.

Explication :

'n' est une expression numérique qui indique le numéro de la fenêtre dont on veut modifier le titre.

La chaîne de caractères *'titre'* devient ce nouveau titre.

Si, avant OPENW, on exécute TITLEW n,"" , on obtient une fenêtre sans ligne de titre. Si on veut obtenir une fenêtre normale avec une ligne pour le titre, il faut entrer TITLEW n," ".

INSTRUCTION **TROFF**

Syntaxe : TROFF

Abrev. : TROF

Exemple :

```
TRON
DO
PRINT I
INC I
IF I=25
  TROFF
ENDIF
LOOP
```

Fonction : Supprime la fonction trace qui a été mise en place avec TRON.

Explication : (pas nécessaire).

INSTRUCTION TRON

Syntaxe : TRON

Abrev. : TR

Exemple :

```
TRON
DO
PRINT I
INC I
IF I=25
  TROFF
ENDIF
LOOP
```

Fonction : Met en place de la fonction trace.

Explication : Affiche la ligne en cours d'exécution.

CONSTANTE **TRUE**

Syntaxe : TRUE

Exemple : Flag!=TRUE

Fonction : Constante -1

Explication :

Il s'agit uniquement d'une autre manière d'écrire (plus lisible) la valeur booléenne 'vrai'.

En outre, dans la version 2.0 du BASIC GfA, les nombres 0, 1, 2 et 3 sont mémorisés dans des cellules-mémoires de 1 octet (au lieu de 7-8 octets). C'est pourquoi les programmes écrits dans la version 1 diminuent de taille, si on effectue la séquence suivante : sauvegarde sur fichier ASCII (avec SAVE,A), NEW, chargement en mémoire de ce fichier avec MERGE, nouvelle sauvegarde avec SAVE. Cela permet en outre d'effacer les noms de variables du fichier SAVE, qui ont été créées lors de l'édition.

FONCTION **TRUNC**

Syntaxe : TRUNC(x)

Exemple :
```
A=3.1415
PRINT TRUNC(A)
PRINT TRUNC(-12)
PRINT TRUNC(-1.99)
```

Fonction : Retourne la partie entière de x obtenue en tronquant la partie après la virgule.

Explication :

'*x*' est une expression numérique quelconque.

Pour des valeurs positives de x, la fonction est identique à INT. La différence apparait pour les valeurs négatives de x.

Exemple : TRUNC(-1.99) donne -1, alors que INT(-1.99) donne -2.

TRUNC (et non pas INT) est le complémentaire de FRAC (voir aussi FRAC). On a donc :

$$x = TRUNC(x) + FRAC(x)$$

La fonction TRUNC est identique à la fonction FIX.

FONCTION TYPE

Syntaxe : TYPE(ptr)

Exemple : (voir *)

Fonction : Fournit le type de la variable qui est pointée.

Explication :

'*ptr*' est une expression entière (normalement de la forme *var).

TYPE(ptr) indique le type de la variable qui est pointée par 'ptr'.

```
0=var
1=var$
2=var%
3=var!
4=var()
5=var$()
6=var%()
7=var!()
```

En cas d'erreur, la fonction retourne la valeur -1.

(voir aussi *).

FONCTION **UPPER$**

Syntaxe : UPPER$(string)

Exemple :
```
A$="basic"
PRINT UPPER$(a$)
PRINT UPPER$("1a")
PRINT UPPER$("GfA")
```

Fonction : Transforme toutes les lettres minuscules (voyelles infléchies incluses) d'une chaîne de caractères en lettres majuscules.

Explication :

'*string*' est une expression alphanumérique quelconque.

Les caractères qui ne sont pas des lettres alphabétiques restent inchangés.

FONCTION **VAL?**

Syntaxe : VAL?(x$)

Exemple :
```
A$="22 DM"
PRINT VAL?(A$)
PRINT VAL?("1.0E+19")
PRINT VAL?("1E19")
PRINT VAL?("&X1010011")
PRINT VAL?("AB123")
```

Fonction : Recherche le nombre maximum de caractères de x$ qui représente une valeur numérique.

Explication :

'*x$*' est une chaîne de caractères quelconque ou une variable alphanumérique.

La fonction *VAL?* recherche la plus grande sous-chaîne de x$ (en partant de la gauche) qui pourrait être convertie avec VAL dans une valeur numérique, et renvoie le nombre de caractères de cette sous-chaîne. Pour une chaîne de caractères qui ne représente aucun nombre ou pour la chaîne vide, on obtient la valeur 0.

L'exemple donne les résultats 2, 7, 4, 9 et 0.

FONCTION **VAL?**

Syntaxe : VAL?(x$)

Exemple :
```
A$="22 DM"
PRINT VAL?(A$)
PRINT VAL?("1.0E+19")
PRINT VAL?("1E19")
PRINT VAL?("&X1010011")
PRINT VAL?("AB123")
```

Fonction : Recherche le nombre maximum de caractères de x$ qui représente une valeur numérique.

Explication :

'x$' est une chaîne de caractères quelconque ou une variable alphanumérique.

La fonction *VAL?* recherche la plus grande sous-chaîne de x$ (en partant de la gauche) qui pourrait être convertie avec VAL dans une valeur numérique, et renvoie le nombre de caractères de cette sous-chaîne. Pour une chaîne de caractères qui ne représente aucun nombre ou pour la chaîne vide, on obtient la valeur 0.

L'exemple donne les résultats 2, 7, 4, 9 et 0.

FONCTION **VARPTR**

Syntaxe : VARPTR(var)

Exemple :
```
A%=17
N$="GfA"
PRINT VARPTR(A%)
PRINT LPEEK(VARPTR(A%))
PRINT VARPTR(N$)
PRINT PEEK(VARPTR(N$))
PRINT PEEK(VARPTR(N$)+1)
PRINT PEEK(VARPTR(N$)+2)
```

Fonction : Détermine l'adresse (de début) d'une variable.

Explication :

'var' est une variable numérique, alphanumérique ou booléenne quelconque. Les variables et leurs contenus sont placés en mémoire selon un schéma bien précis (voir annexe D : organisation des variables).

VARPTR(var) détermine l'adresse du premier octet où est mémorisé 'var'.

Ainsi, avec l'instruction VARPTR(A%) tirée de l'exemple, on obtient l'adresse du premier des quatre octets qui contiennent A%.

PEEK(VARPTR(N$)) (également tiré de l'exemple ci-dessus) renvoie le code ASCII du premier caractère de la chaîne N$, à savoir 71.

FONCTION **VDIBASE**

Syntaxe : VDIBASE

Exemple : (pas nécessaire).

Fonction : Pour s'amuser en POKant !

Explication :

Cette fonction permet d'obtenir l'adresse située au-delà du BASIC GfA et des variables et tables utilisées. A partir de cette adresse, on trouve les paramètres-VDI du GEM (type d'écriture, clipping, etc...), et enfin le programme écrit en BASIC. Si on Peeke ou Poke dans cette zone-mémoire, on peut obtenir divers effets (également de méchants plantages). Il vaut donc mieux ne rien avoir d'important en mémoire si l'on veut modifier le GEM sans risques.

INSTRUCTION VOID

Syntaxe : VOID expression

Abrev. : VO

Exemple : VOID FRE(0)
VOID INP(2)

Fonction : Remplace **DUMMY =** (plus rapide).

Explication :

Cette instruction exécute un calcul sans mémoriser le résultat. Cela peut sembler inutile mais cette commande est très pratique pour certaines applications. En particulier, le Garbage-collection (FRE(0)), l'attente de la pression d'une touche (INP(2)), l'appel de différentes routines, BIOS, XBIOS, GEMDOS et C: qui ne retournent aucun paramètre ou dont on n'utilise pas la valeur-retour.

INSTRUCTION **VSYNC**

Suntaxe : **VSYNC**

Abrev. : VS

Fonction : Synchronisation avec le balayage de l'écran.

Explication :

Cette instruction arrête l'exécution du programme jusqu'à ce qu'une impulsion de synchronisation soit émise, c'est-à-dire, jusqu'à ce que l'image soit totalement affichée à l'écran. Cela permet d'éviter les parasites ou le tremblement de l'image lorsque l'on fait une animation (par exemple avec GET/PUT). Il faut que le dessin se fasse très rapidement ou bien qu'il soit affiché après chaque balayage de ligne. Cela correspond à la fonction VOID XBIOS(37).

INSTRUCTION WAVE

Syntaxe : WAVE canal,enveloppe,forme,période,durée

Abrev. : WA

Exemple :
```
SOUND 1,15,1,4,20
SOUND 2,15,4,4,20
SOUND 3,15,8,4,20
WAVE 7,7,0,65535,300
WAVE 0,0
```

Fonction : Permet de combiner les trois canaux et de gênérer du bruit.

Explication :

'canal', *'enveloppe'*, *'forme'*, *'période'* et *'durée'* sont des expressions entières.

'canal' est évalué bit par bit. Pour combiner les différents canaux, il suffit d'additionner les valeurs des bits correspondantes :

1 = canal 1
2 = canal 2
4 = canal 4
8 = bruits sur canal 1
16 = bruits sur canal 2
32 = bruits sur canal 3

A cela, on peut encore ajouter 256 fois la période du générateur de bruits (0 à 31).

'enveloppe' détermine, selon les valeurs des trois premiers bits, quels sont les canaux qui utilisent la courbe enveloppe (ex. : canaux 1 et 3 pour 5=&X101). ==>

INSTRUCTION	WAVE suite

'*forme*' détermine la forme de la courbe enveloppe :

0-3	=	linéaire décroissant
4-7	=	linéaire croissant, puis retombant net
8	=	dents de scie décroissantes
9	=	linéaire décroissant (comme 0-3)
10	=	triangle, décroissant au début
11	=	linéaire décroissant, puis saut à une valeur haute
12	=	dents de scie croissantes
13	=	linéaire croissant, constant
14	=	triangle, croissant au début
15	=	linéaire croissant, puis retombant (comme 4-7)

'*periode*' détermine la période de la courbe enveloppe (plus la période est grande, plus la courbe enveloppe est allongée).

'*durée*' est le temps, mesurée en 1/50 secondes, qui doit s'écouler avant d'exécuter l'instruction suivante.

Les paramètres qui se trouvent à la fin de l'instruction et qui restent inchangés peuvent être omis.

WAVE 0,0 déconnecte tous les canaux sonores.

INSTRUCTION DE STRUCTURE WHILE ... WEND

Syntaxe :
```
WHILE condition
WEND
```

Abrev. :
```
W
WE
```

Exemple :
```
WHILE A<10
   A=A+1
   PRINT A
WEND
```

Fonction : Met en place une boucle conditionnelle.

Explication :

Le bloc d'instructions entre WHILE et WEND est exécuté (éventuellement plusieurs fois) tant que la condition est vérifiée. Comme on teste la condition au début de l'instruction, il est fort possible que la boucle ne soit parcourue aucune fois, à l'inverse de l'instruction REPEAT...UNTIL.

INSTRUCTION I/O — **WRITE**
WRITE #

Syntaxe :
WRITE [expressions][;]
WRITE #n[,expressions][;]

Abrev. :
WR

Exemple :
```
OPEN "O",#1,"DAT"
WRITE #1,"Dupont","Marcel",18
CLOSE #1
OPEN "I",#1,"DAT"
INPUT #1,NOM$,PRENOM$,AGE
PRINT PRENOM$'NOM$'
PRINT "est agé de"'AGE'"ans."
CLOSE #1
```

Fonction :
Permet de sauvegarder, sur un fichier séquentiel, des données que l'on peut relire avec INPUT.

Explication :

'expressions' est une liste d'expressions séparées par des virgules.

Contrairement à l'instruction PRINT, les expressions sont séparées par le caractère ',' et les chaînes de caractères sont comprises entre guillemets. Ce format est identique à celui de l'instruction INPUT.

(voir aussi PRINT).

INSTRUCTION/FONCTION *

Syntaxe : *

Exemple :

```
DIM A$(9)
FOR I=0 TO 9
  A$(I)=STR$(RND)
NEXT I
@DSP
@SORT(*A$())
PRINT "Résultat après un tri"
@DSP
PROCEDURE DSP
  FOR i=0 TO 9
    PRINT a$(i)
  NEXT i
RETURN
PROCEDURE SORT(p.sarr)    !Tri x$()
  IF TYPE(p.sarr)<>5
    ERROR 47        !paramètre doit être une chaîne
  ENDIF
  SWAP *p.sarr,x$()        !Tableau de travail
  LOCAL i%,j%
  ' tri bulle - lent
  FOR i%=DIM?(x$())-2 DOWNTO 0
    FOR j%=0 to i%
      IF X$(j%)>x$(j%+1)
        SWAP X$(j%),X$(j%+1)
      ENDIF
    NEXT j%
  NEXT i%
  SWAP *p.sarr,x$()   !Retour du tableau
RETURN
```

Fonction : Passage indirect de variables et de tableaux. ==>

INSTRUCTION/FONCTION •
suite

Explication :

L'étoile ne sert pas seulement pour la multiplication. En effet, elle désigne aussi l'adressage indirect pour les procédures avec retour de paramètres ou avec transmission de tableaux.

GOSUB xxx(*a) ne transmet pas le contenu de la variable a, mais son adresse.

Inversement, **ptr=x* modifie la variable dont l'adresse est ptr. En ce qui concerne les tableaux, la seule possibilité pour lire ou modifier le contenu des différents éléments est d'utiliser l'instruction SWAP. Avec TYPE(), il est possible de déterminer de quel type est la variable qui est pointée.

```
ptr%=*a
*ptr%=17                        !a=17

ptr%=*a%
*ptr%=17                        !a%=17

ptr%=*a$
*ptr%="Test"                    !a$="Test"

ptr%=*A()
swap *ptr%,x()
dim x(9)                        !dim a(9)
swap *ptr%,x()

@sum(*a,7,3)                    !A=7+3
procedure sum(p.num%,x,y)
  *p.num%=x+y
return
```

INSTRUCTION/FONCTION • **suite**

Important : Les variables locales et variables globales de même nom sont placées à la même adresse par l'interpréteur (les contenus des variables globales sont sauvegardés lors de l'apparition de l'instruction LOCAL, et à nouveau restaurés après l'instruction RETURN).
C'est pourquoi aucune variable locale ne peut avoir le même nom qu'un paramètre devant être retourné.

Lors d'une mauvaise utilisation des pointeurs, il apparait un message d'erreur (voir TYPE).

FONCTION ==

Syntaxe : a==b

Exemple :
```
FOR I=-1 TO 1 STEP 0.1
  EXIT IF I==0.5
NEXT I
PRINT I
```

Fonction : Opérateur de comparaison : environ égal à.

Explication :

'a' et 'b' sont des expressions numériques. L'opérateur == doit être appliqué de la même manière que l'opérateur =. Cependant, on ne teste pas l'égalité parfaite des deux expressions, mais seulement l'égalité approchée (28 bits de la mantisse sont comparés, c-à-d environ 8.5 chiffres). Dans l'exemple ci-dessus, on sort de boucle, bien que la variable I n'atteigne certainement pas exactement la valeur 0.5 à cause des erreurs d'arrondi.

(voir aussi DEFNUM et PRINT USING).

ANNEXE B.

Messages d'erreurs

MESSAGES D'ERREURS DU BASIC GfA

0 = Division par zéro
1 = Dépassement de capacité
2 = Le nombre n'est pas un Integer -2147483648 .. 2147483647
3 = Le nombre n'est pas un octet 0 .. 255
4 = Le nombre n'est pas un mot 0 .. 65535
5 = Racine carrée d'un nombre négatif impossible
6 = Logarithme d'un nombre inférieur à zéro impossible
7 = Erreur inconnue
8 = Mémoire pleine
9 = Fonction ou instruction impossible
10 = Chaîne trop longue, max 32767 caractères
11 = Le programme n'est pas en GfA BASIC version 2.0
12 = Programme trop grand, mémoire pleine
13 = Le fichier programme n'est pas en GfA BASIC
14 = Champ dimensionné deux fois
15 = Champ non dimensionné
16 = Index de champ trop grand
17 = Index de dim trop grand
18 = Mauvais nombre d'indices
19 = Procédure introuvable
20 = Label introuvable
21 = Pour OPEN, utiliser 'I'nput, 'O'utput, 'R'andom, 'A'ppend, 'U'pdate
22 = Fichier déjà ouvert
23 = Mauvais numéro de fichier
24 = Fichier non ouvert
25 = Mauvaise saisie, ce n'est pas un nombre
26 = Fin de fichier atteinte , EOF
27 = Trop de points pour Polyline/Polyfill, max 128

28 = Le champ ne peut avoir qu'une dimension
29 = Nombre de points plus grand que le champ
30 = Merge, ce n'est pas un fichier ASCII
31 = Merge, ligne trop longue
32 = ==> Syntaxe incorrecte, arrêt du programme
33 = Marque non définie
34 = Trop peu de données
35 = Donnée non numérique
36 = Erreur de syntaxe dans la donnée, utiliser les "" par paire
37 = Disquette pleine
38 = Instruction impossible en mode direct
39 = Erreur de programmation, GOSUB impossible
40 = CLEAR n'est pas possible dans une boucle FOR NEXT ou une procédure.
41 = CONT impossible
42 = Trop peu de paramètres
43 = Expression trop complexe
44 = Fonction indéfinie
45 = Trop de paramètres
46 = Paramètre inexact, ce doit être un nombre
47 = Paramètre inexact, ce doit être une chaîne
48 = OPEN "R", enregistrement trop long
49 = Trop de fichiers "R" (max 10)
50 = Pas de fichier "R"
51 = Seul un champ est possible avec un OPEN "R"
52 = Champ plus grand que l'enregistrement
53 = Trop de champs (max 19)
54 = Mauvaise longueur d'enregistrement GET/PUT
55 = Mauvais numéro d'enregistrement GET/PUT
60 = Longueur de chaîne de SPRITE erronée
62 = Erreur dans MENU
63 = Erreur dans RESERVE
64 = Erreur dans Pointeur
90 = Erreur dans LOCAL
91 = Erreur dans FOR
92 = RESUME (NEXT) impossible, FATAL, FOR ou LOCAL
100 = (c) Copyright 1986, GfA Systemtechnik

MESSAGES D'ERREURS BOMBE

102 = 2 bombes - erreur Bus, peut-être mauvais peek ou poke
103 = 3 bombes - erreur d'adresse, adresse de mot impaire
Avec Dpoke, Dpeek, Lpoke ou Lpeek ?
104 = 4 bombes - exécution d'une instruction 68000
ne convenant pas
105 = 5 bombes - Division par zéro en langage machine 68000
106 = 6 bombes - exception CHK, interruption du 68000 par
instruction CHK
107 = 7 bombes - exeption TRAPV, interruption du 68000 par
instruction TRAPV
108 = 8 bombes - interruption 68000 par exécution
d'une instruction privilégiée
109 = 9 bombes - exception trace, interruption TRACE avec 68000

MESSAGES D'ERREUR DU TOS

-1 = Erreur générale
-2 = Drive not Ready, désynchronisation
-3 = Instruction inconnue
-4 = Erreur CRC, test de somme du disque incorrecte
-5 = Bad request, instruction ne convenant pas
-6 = Seek error, piste introuvable
-7 = Unknown media, mauvais bootsector
-8 = Secteur introuvable
-9 = Pas de papier
-10 = Erreur d'écriture
-11 = Erreur de lecture
-12 = Erreur générale 12
-13 = Disquette protégée
-14 = Vous avez changé de disquette
-15 = Appareil inconnu
-16 = Mauvais secteur (verify)
-17 = Insérer une autre disquette
-32 = Numéro de fonction incorrect
-33 = Fichier introuvable

-34 = Nom de PATH introuvable, chemin dans directory
-35 = Trop de fichiers ouverts
-36 = Accès impossible
-37 = Handle incorrect
-39 = Mémoire pleine
-40 = Adresse de bloc mémoire incorrecte
-46 = Numéro de lecteur incorrect
-49 = Il n'y a pas d'autres données
-64 = Erreur GEMDOS, seek incorrect
-65 = Erreur interne de GEMDOS
-66 = Ce n'est pas un fichier binaire
-67 = Erreur de bloc mémoire

MESSAGE D'ERREUR DE L'EDITEUR

While sans Wend
Repeat sans Until
Do sans Loop
For sans Next
Wend sans While
Until sans Repeat
Loop sans Do
Next sans For
If sans Endif
Endif sans If
Else sans If
Else sans Endif
Exit sans boucle
Procédure sans Return
Procédure dans boucle
Procédure définie deux fois
Return sans procédure
Marque définie deux fois
Local autorisé seulement dans une procédure
Local interdit dans une boucle
Fonction définie deux fois
Goto dans/vers extérieur une boucle For/Next ou une Procédure

Resume dans boucle For-Next
Resume sans Procédure
Erreur de syntaxe
Ligne trop longue

ANNEXE C.

Le code ASCII

La table suivante contient les différents codes ASCII et leurs caractères correspondants.

Les caractères dont les codes ASCII sont compris entre 32 et 255 peuvent être affichés à l'écran en utilisant simplement l'instruction CHR$(x).

Les caractères dont les codes sont compris entre 0 et 31 sont des caractères de contrôle. Si on applique l'instruction CHR$ à ces caractères, ceux-ci ne sont non pas affichés mais exécutés.

Tous les caractères peuvent être entrés au clavier de différentes façons :

* La plupart des caractères de contrôle en appuyant simultanément sur la touche <CONTROL> et une autre touche.

* Beaucoup de caractères spéciaux en appuyant simultanément sur la touche <ALTERNATE> et une autre touche.

* La plupart des caractères spéciaux en enfonçant simultanément les touches <CONTROL> et <S>, puis une touche quelconque. (Ce faisant, on affiche le caractère dont le code ASCII est égal au code ASCII du caractère entré augmenté de 128.)

* Presque tous les caractères (sauf les deux caractères correspondants aux codes 13 et 0) en appuyant simultanément sur les touches <CONTROL> et <A> puis en entrant le code ASCII du caractère à afficher.

A ce sujet, l'introduction d'un caractère doit être validée avec une touche quelconque, si le code ASCII est plus petit que 26.

Table des codes ASCII

ASCII Valeur	Caractères		ASCII Valeur	Caractères
0		NUL	32	
1	☺	SOH	33	!
2	☻	STX	34	"
3	♥	ETX	35	#
4	♦	EOT	36	$
5	♣	ENQ	37	%
6	♠	ACK	38	&
7	•	BEL	39	'
8	◘	BS	40	(
9	○	HT	41	)
10	◙	LF	42	*
11	♂	VT	43	+
12	♀	FF	44	,
13	♪	CR	45	-
14	♫	SO	46	.
15	☼	SI	47	/
16	►	DLE	48	0
17	◄	DC1	49	1
18	↕	DC2	50	2
19	‼	DC3	51	3
20	¶	DC4	52	4
21	§	NAK	53	5
22	▬	SYN	54	6
23	↨	ETB	55	7
24	↑	CAN	56	8
25	↓	EM	57	9
26	→	SUB	58	:
27	←	ESC	59	;
28	∟	FS	60	<
29	↔	GS	61	=
30	▲	RS	62	>
31	▼	US	63	?

ASCII Valeur	Caractères	ASCII Valeur	Caractères
64	@	96	`
65	A	97	a
66	B	98	b
67	C	99	c
68	D	100	d
69	E	101	e
70	F	102	f
71	G	103	g
72	H	104	h
73	I	105	i
74	J	106	j
75	K	107	k
76	L	108	l
77	M	109	m
78	N	110	n
79	O	111	o
80	P	112	p
81	Q	113	q
82	R	114	r
83	S	115	s
84	T	116	t
85	U	117	u
86	V	118	v
87	W	119	w
88	X	120	x
89	Y	121	y
90	Z	122	z
91	[	123	{
92	\	124	|
93	]	125	}
94	^	126	~
95	_	127	Δ

ASCII Valeurs	Caractères	ASCII Valeurs	Caractères
128	Ç	160	á
129	ü	161	í
130	é	162	ó
131	â	163	ú
132	ä	164	ñ
133	à	165	Ñ
134	å	166	ª
135	ç	167	º
136	ê	168	¿
137	ë	169	⌐
138	è	170	¬
139	ï	171	½
140	î	172	¼
141	ì	173	¡
142	Ä	174	«
143	Å	175	»
144	É	176	ã
145	æ	177	õ
146	Æ	178	Ø
147	ô	179	ø
148	ö	180	Œ
149	ò	181	œ
150	û	182	Ã
151	ù	183	Õ
152	ÿ	184	¨
153	Ö	185	´
154	Ü	186	†
155	¢	187	¶
156	£	188	©
157	¥	189	®
158	ß	190	™
159	ƒ	191	

ASCII Valeurs	Caractères	ASCII Valeurs	Caractères
192	ij	224	α
193	ij	225	ß
194	א	226	Γ
195	ב	227	π
196	ג	228	Σ
197	ד	229	σ
198	ה	230	µ
199	ו	231	τ
200	ז	232	Φ
201	ח	233	Θ
202	ט	234	Ω
203	י	235	δ
204	כ	236	∞
205	ל	237	φ
206	מ	238	ε
207	נ	239	∩
208	ס	240	≡
209	ע	241	±
210	פ	242	≥
211	צ	243	≤
212	ק	244	⌠
213	ר	245	⌡
214	ש	246	÷
215	ת	247	≈
216	ו	248	°
217	ך	249	∙
218	ם	250	·
219	ן	251	√
220	ף	252	ⁿ
221	₪	253	²
222	^	254	³
223	∞	255	¯

ANNEXE D.

Les variables et leur organisation

Le GfA BASIC possède quatre types de variables différents :

* **Les variables sans suffixe distinctif** représentent des réels à virgule flottante qui sont mémorisés sur 6 octets sous format binaire. La précision est de 11 chiffres significatifs. Dans la représentation normale (par ex. 3.546E+17), l'exposant ne peut pas dépasser la valeur 154.

* **Les variables avec le suffixe %** (par exemple A%) sont des variables entières (Integer), dont la mémorisation nécessite 4 octets. Avec ce type de variables, on peut représenter des nombres entiers allant de -2 milliards à +2 milliards environ (-2147483648 à +2147483647 exactement).

* **Les variables booléennes** sont suivies du caractère ! et ne peuvent prendre que deux valeurs : la valeur 0 (pour Faux) et -1 (en fait toute valeur différentes de 0 : pour Vrai). Elles utilisent deux octets.

* **Les variables alphanumériques** (variables chaînes de caractères) ont pour suffixe le caractère $. De telles variables peuvent contenir au maximum 32767 caractères, car, dans le descripteur (voir plus bas) d'une variable alphanumérique, seuls deux octets sont utilisés pour indiquer la longueur.

Les noms de variables doivent toujours commencer par une lettre alphabétique et peuvent avoir une longueur quelconque (limitée bien sûr par la taille d'une ligne).

Pour différencier les variables, toute la longueur du nom est prise en compte.

Les caractères suivant la lettre alphabétique initiale peuvent être des lettres, des chiffres, le souligné ou le point.

Si une instruction d'affectation commence par LET, on peut même utiliser des mots-clés (noms d'instructions) comme noms de variables. Ainsi, la ligne suivante est tout à fait autorisée :

```
LET LET=17.
```

En BASIC GfA, il y a donc très peu de mots réservés.

Grâce à l'instruction VARPTR(var) (pointeur de variables), on peut accéder à l'adresse en mémoire principale du premier octet de la zone-mémoire où le BASIC GfA place le contenu de 'var'.

Pour pouvoir mémoriser les chaînes de caractères et les tableaux (champs), on utilise un descripteur.

Le descripteur est stocké sur 6 octets, dont les quatre premiers représentent une adresse. La signification des deux derniers est différente selon qu'il s'agit d'une chaîne de caractères ou d'un tableau.

Examinons donc séparément les descripteurs de chaînes de caractères et de tableaux. Commençons par le plus simple : l'organisation des chaînes de caractères en mémoire :

Avec l'instruction ARRPTR(var), on obtient l'adresse du premier octet du descripteur. Comme nous l'avons déjà dit, les quatre premiers octets du descripteur indiquent une adresse. C'est l'adresse où est placé le premier caractère de la chaîne. Cette adresse est identique à celle que retourne l'instruction VARPTR. Les deux derniers octets du descripteur indique la taille de la chaîne. A la fin de la chaîne mémorisée, on trouve, derrière un éventuel octet de remplissage (placé au cas où la taille de la chaîne est impaire), l'adresse de début du descripteur correspondant (on appelle cette adresse 'backtrailer'.

Il permet d'accélerer sensiblement l'exécution du processus appelé Garbage-Collection (la collecte et l'effacement des zones-mémoires qui ne sont plus utilisées)).

Le programme suivant devrait permettre de comprendre encore mieux l'organisation des chaînes de caractères en mémoire :

```
A$="GfA"
PRINT "Adresse du descripteur:"
PRINT ARRPTR(A$)
PRINT "Adresse de début de la chaîne:"
PRINT LPEEK(ARRPTR(A$))
PRINT "Pour vérification:"
PRINT VARPTR(A$)
PRINT "Longueur de la chaîne:"
PRINT DPEEK(ARRPTR(A$)+4)
PRINT "Chaîne de caractères:"
PRINT CHR$(PEEK(VARPTR(A$)));
PRINT CHR$(PEEK(VARPTR(A$)+1));
PRINT CHR$(PEEK(VARPTR(A$)+2))
PRINT "Octet de remplissage:"
PRINT PEEK(VARPTR(A$)+3)
PRINT "Backtrailer:"
PRINT LPEEK(VARPTR(A$)+4)
```

Examinons à présent la manière dont les tableaux (champs) sont stockés en mémoire :

On accède aux tableaux de la même façon que pour les chaînes, c'est-à-dire par l'intermédiaire d'un descripteur (descripteur de tableau). L'adresse de début du descripteur est également obtenu à l'aide de l'instruction ARRPTR(tab). Les quatre premiers octets du descripteur de tableau contiennent l'adresse du début du tableau. Les deux derniers octets indiquent le nombre de dimensions du tableau (exemple : quatre pour A(3,4,5,7).

Au début du tableau se trouvent, contenu dans chaque fois quatre octets, les nombres d'éléments dans chaque dimension. On commence par la dernière dimension (avec l'exemple précédent A(3,5,5,7), les quatre premiers octets contiennent la valeur 8, les quatre suivants la valeur 6, les quatre suivants la valeur 5 et les quatre derniers la valeur 4 (ne pas oublier que l'indice commence à 0)).

On trouve ensuite le contenu des différents éléments du tableau. Pour des tableaux de réels, chaque élément occupe 6 octets, pour des tableaux d'entiers relatifs (Integer), 4 octets, et pour des tableaux de booléens, 1 seul bit.

S'il s'agit d'un tableau de chaînes de caractères, on n'obtient pas directement le contenu des chaînes, mais leurs descripteurs. Ces descripteurs de chaînes ont le même format et la même fonction que ceux décrits dans la première partie.

Pour se familiariser avec cette organisation des tableaux qui peut paraître confuse au premier coup d'oeil (surtout en ce qui concerne les tableaux de chaînes de caractères), il est conseillé de programmer l'exemple suivant :

```
DIM A$(1,0)
A$(0,0)="A"
A$(1,0)="BC"
D=ARRPTR(A$())
PRINT "Adresse du descripteur de tableau:"D
F=LPEEK(D)
PRINT "Adresse du tableau:"F
PRINT "Nombre de dimension:"
PRINT DPEEK(D+4)
PRINT "Nombre d'éléments dans la 2ème dim.:"
PRINT LPEEK(F)
PRINT "Nombre d'éléments dans la 1ère dim.:"
PRINT LPEEK(F+4)
PRINT "Adr. du descripteur pour 0,0:"
PRINT F+8
PRINT "Adr. dans le descripteur pour 0,0:"
PRINT LPEEK(F+8)
PRINT "Longueur de la chaîne 0,0:"
PRINT DPEEK(F+12)
PRINT "Adr. du descripteur pour 1,0:"
PRINT F+14
PRINT "Adr. dans le descripteur pour 1,0:"
PRINT LPEEK(F+14)
PRINT "Longueur de la chaîne 1,0:"
PRINT DPEEK(F+18)
PRINT "Vérification pour 0,0:"
PRINT VARPTR(A$(0,0))
PRINT "Vérification pour 1,0:"
PRINT VARPTR(A$(1,0))
PRINT "Chaîne 0,0:"
PRINT CHR$(PEEK(LPEEK(F+8)))
PRINT "Chaîne 1,0:"
PRINT CHR$(PEEK(LPEEK(F+14)));
PRINT CHR$(PEEK(LPEEK(F+14)+1))
PRINT "Backtrailer pour 0,0:"
PRINT LPEEK(LPEEK(F+8)+2)
PRINT "Backtrailer pour 1,0:"
PRINT LPEEK(LPEEK(F+14)+2)
```

ANNEXE E.

Fonctions spéciales / Divers

Cette partie donne un bref aperçu des appels du VDI et de l'AES, ainsi que des fonctions OS spéciales.

ADDRIN	:	Adresse du bloc AES-Adress-Input
ADDROUT	:	Adresse du bloc AES-Adress-Output
CONTRL	:	Adresse du bloc VDI-Control
GB	:	Adresse du bloc AES-Parameter
GCONTRL	:	Adresse du bloc AES-Control
GINTIN	:	Adresse du bloc AES-Integer-Input
GINOUT	:	Adresse du bloc AES-Integer-Output
INTIN	:	Adresse du bloc VDI-Integer-Input
INTOUT	:	Adresse du bloc VDI-Integer-Output
PTSIN	:	Adresse du bloc VDI-Point-Input
PTSOUT	:	Adresse du bloc VDI-Point-Output

L'écriture et la lecture à l'intérieur de ces blocs est possible à l'aide des instructions DPOKE, DPEEK, LPOKE et LPEEK.

Pour appeler le GEM, on utilise l'instruction

GEMSYS : il y a deux appels différents possibles :

1. GEMSYS n ou GEMSYS (n)
L'AES est appelé après avoir défini le bloc GCONTRL.

2. GEMSYS
L'AES est appelé sans modification du bloc GCONTRL.

Pour appeler le VDI, on utilise l'instruction :

VDISYS : il y a également deux possibilités :

1. VDISYS n ou VDISYS (n)
Le VDI est appelé après avoir introduit le numéro de la fonction dans le bloc CONTRL.

2. VDISYS
Le VDI est appelé sans modification du numéro de fonction dans le bloc CONTRL.

WINDTAB : Adresse de la table des paramètres des fenêtres.

Dans cette table, on trouve différents paramètres, tous de la taille d'un mot (16 bits), qui sont nécessaires à la gestion des fenêtres. Grâce à une programmation astucieuse, on peut utiliser ces paramètres. La table est construite de la manière suivante :

1.1 Handle de la fenêtre ou zéro
  2 Attribut de la fenêtre
  3 Abscisse X de la fenêtre
  4 Ordonnée Y de la fenêtre
  5 Largeur de la fenêtre
  6 Hauteur de la fenêtre
2. comme 1.
3. comme 1.
4. comme 1.
5.1 Paramètre Dummy (-1)
  2 Paramètre Dummy (0)
  3 Abscisse X de l'écran
  4 Ordonnée Y de l'écran
  5 Largeur de l'écran
  6 Hauteur de l'écran
6.1 Abscisse X du point d'intersection des 4 fenêtres
  2 Ordonnée Y du point d'intersection des 4 fenêtres
7.1 Abscisse X de l'origine
  2 Ordonnée Y de l'origine

Fonctions OS spéciales.

BIOS : Bios(f [,listeparam])
XBIOS : Xbios(f [,listeparam])
GEMDOS : Gemdos(f [,listeparam])

Ces fonctions appellent des routines du TOS (appelé aussi Gemdos), le système d'exploitation spécifique au ST. (Le GEM ne représente que l'interface utilisateur.)

Explication de la syntaxe des fonctions :

1. Ce sont des fonctions qui retournent un paramètre de 32 bits (comme habituellement en C).

2. f est le numéro de fonction.

3. Ce qui suit dépend de la fonction. Beaucoup d'instructions nécessitent des paramètres qui sont soit des mots (16 bits), soit des mots longs (32 bits).

   Exemples : a=XBIOS(20) ⟶ Hardcopy

   ```
   a$=SPACE$(512)
   a=BIOS(4,0,L:VARPTR(A$),1,1,0)
   ```

   Lit 1 secteur, à partir du secteur logique 1 de l'unité 0 (A:) et le mémorise dans la variable alphanumérique AS.

   ```
   Adresseecran = XBIOS(2)
   ```

   Détermine l'adresse de début de la mémoire-écran.

Lancement de l'interpréteur RUN-ONLY

Pour exécuter un programme avec l'interpréteur RUN-ONLY, il y a deux possibilités.

1. Vous lancez l'exécution du programme 'GFABASRO.PRG' avec un double-click. Il apparait maintenant une boîte de sélection de fichiers. Vous choisissez et lancez ensuite le programme souhaité.

2. Si vous voulez appeler un programme avec l'interpréteur RUN-ONLY directement à partir de l'environnement GEM, vous devez effectuer la séquence suivante :

   Vous cliquez tout d'abord le programme 'GFABASRO.PRG'. Vous choisissez ensuite l'option 'Installer une application" du menu 'Options'. Vous modifiez maintenant le 'Type de document' en 'BAS'. Vérifiez que le type d'application reste bien 'GEM'.

   Ensuite, vous cliquez le bouton 'Confirmer'. Enfin, vous devez sauvegarder avec l'option 'Sauvegarder le bureau' cette configuration sur une disquette système.

   Remarque : Cette procédure peut aussi être effectuée avec "GFABASIC".

Pour enlever les numéros de lignes des programmes en BASIC ST :

Sur la disquette d'origine du BASIC GfA se trouve le programme 'CONVERT.BAS' écrit en BASIC GfA. Avec ce programme, il est possible d'enlever les numéros de lignes pour des programmes écrits en BASIC ST et de transformer les instructions chaînées sur une même ligne en plusieurs lignes d'une seule instruction chaque fois. Le programme transformé doit être chargé dans l'éditeur avec MERGE. Ce programme s'explique de lui-même et ne nécessite donc pas de description.

De par sa puissance et sa convivialité, GFA s'est imposé comme "le" langage sur ATARI ST. Aujourd'hui GFA surpasse GFA avec la version 3.0 dont la richesse et l'environnement renforce encore sa suprématie.

GFA BASIC 3.0

Totalement réécrit, avec des performances considérablement accrues, le GFA Basic 3.0 surpasse tous les Basic conçus pour l'ATARI ST. L'éditeur a été modifié: l'écran principal affiche désormais de nouveaux symboles: heure en temps réel (modifiable dans la barre de menu), compteur de ligne asservi au curseur, témoin de verrouillage des majuscules et du bloc numérique. De plus, un petit sigle ATARI permet par simple clic d'accéder aux accessoires de bureau de GEM sans quitter l'application en cours. Dans ce mode vous définirez 4 formats différents pour les mots clés et variables de l'éditeur: suffixes optionnels pour les variables, première lettre en capitale pour les mots clés et variables... Afin de gagner du temps dans les développements, de nouveaux outils vous faciliteront la tâche: le bloc numérique permet le déplacement du curseur, l'appel des fonctions peut être effectué entre autre par la combinaison ALT/bloc numérique. 200 nouvelles fonctions: fonctions mathématiques, opérations sur les bits, interruptions, nouveaux types de variables... Les nouvelles fonctions liées à la définition des procédures ou des variables font du GFA 3.0 un langage structuré sans en subir les contraintes. Des excès de vitesse: pour certaines instructions comme PSET, FOR... NEXT ou INPUT le gain de temps va de 100 à 400%! En moyenne, le GFA 3.0 non compilé tourne 50% plus vite que son prédécesseur. Réf. ST 027. 750 F.

PROGRAMMATION EN GFA BASIC 3.0

C'est le complément indispensable pour réaliser des applications poussées capables de tirer parti des fabuleuses possibilités de la version 3.0. Cette bible du programmeur exploite les nouvelles instructions du langage telles que: programmation structurée, nouveaux types de variable, instructions Line-A, routines assembleur, bibliothèque AES... La disquette offre de très nombreux exemples et une dizaine de programmes utilitaires allant de la sortie de textes en assembleur à la création d'icônes. Réf. ML 638. 349 F avec la disquette. ▲

GFA BASIC 2.0

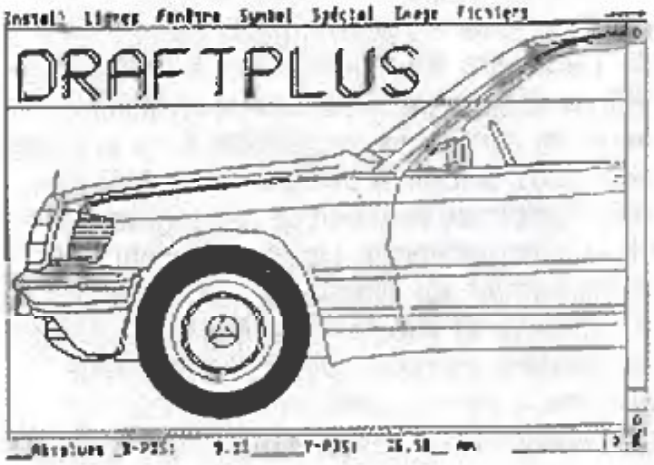
Le GFA Basic s'est imposé comme le langage standard sur ST. Rapide, puissant et simple, il représente plus de 200 commandes pour programmer la souris, les zones d'alerte, les menus déroulants... Ses caractéristiques: programmation structurée, interface GEM ultra rapide, interpréteur compact, éditeur puissant, calcul sur 11 chiffres... Instructions: REPEAT, UNTIL, DO, LOOP, CIRCLE, FILL, TIMER, XBIOS... Bien sûr, les programmes créés en GFA Basic peuvent être compilés avec GFA COMPILATEUR. Réf. ST 012. 495 F.

GFA COMPILATEUR 2.0

Pour accélérer vos programmes en GFA Basic au niveau du C, Turbo Pascal ou Assembleur, une seule solution: le GFA COMPILATEUR. Il décuple la vitesse des applications quasi instantanément, sans passer par le Run Time ou autres librairies de programmes, sans aucun LINKER, en générant du code machine compact et très performant. Simple d'utilisation, il vous fera bénéficier de la formidable rapidité du ST (ne compile que le GFA Basic). Réf. ST 013. 295 F.

GFA DRAFT PLUS

GFA DRAFT PLUS est un programme professionnel de CAO en deux dimensions permettant de réaliser schémas de connexion, dessins techniques, croquis ou plans. Capacités: création de dessins sur 255 plans; lignes variant de 0.3 à 4.5 mm en noir ou en couleur (sur imprimante, écran ou traceur)... GFA DRAFT PLUS possède des guides-lignes (aimantation), une grille de positionnement variable, de nombreuses fonctions: motifs, hachures, agrandissement, déformation, rotation, zoom, symboles liés aux touches de fonctions. Réf. ST 017. 990 F.



GFA ASSEMBLEUR

Programmes de jeu, animations graphiques hyper rapides... GFA ASSEMBLEUR a été conçu pour satisfaire tous les développeurs. L'éditeur corrige instantanément les erreurs. Accessible avec la souris, il permet la recherche et le remplacement de zones programmables, de nombreux réglage et de très nombreux raccourcis au clavier. Il offre une fonction très appréciable d'autocorrection pendant la saisie qui évite les mauvaises surprises à l'exécution. L'assemblage est conditionné et récursif et possède une liste impressionnante de caractéristiques comme le compactage des fichiers pour un gain de place appréciable. Avec son débugger le GFA ASSEMBLEUR est un outil unique! Réf. ST 033. 750 F. (à paraître novembre).

L'ENERGIE MICRO

SUPERBASE 2: LE SGBD VEDETTE

Avec SUPERBASE 2, la gestion des données devient un jeu d'enfant. Son extrême convivialité, son indéniable puissance permettent à ce SGBD de répondre à la plupart de vos besoins, dans les domaines d'application les ▶ plus variés. Son concept: permettre à quiconque, sans connaissances préalables, d'élaborer une gestion de fichiers personnalisée, par le simple emploi de la souris et des outils de GEM. Sa force: une organisation relationnelle multi-fichiers, capable de réaliser des états complexes. Ses capacités graphiques permettent l'importation d'images, de tableaux, de graphiques pour donner à vos documents une qualité parfaite. Enfin, SUPERBASE 2 est doté d'un éditeur de texte fonctionnel comprenant les fonctions essentielles: modification de marge, tabulations, textes en gras, italique, soulignés... Idéal pour le mailing, vous pourrez par exemple personnaliser vos lettres et rapports à partir des informations de la base de données ou inversement, commenter vos masques de saisies ou vos états à partir de fichiers texte. ATARI ST: Réf. ST 032. AMIGA: Réf. MA 366. 990 F. (à paraître octobre).

FACILITÉ D'UTILISATION

- Commande type "magnétoscope"
- Sélection instantanée grâce à la sourie.
- Entièrement contrôle par des menus déroulants.
- Fonction dédiée à l'impression d'étiquettes.
- Sauvegarde des formats d'impression pour utilisation répétée.
- Grand écran de travail permettant des saisies personnalisées.
- Echange de données avec d'autres logiciels.
- Modification de la structure des fichiers sans altérer les enregistrements.

CAPACITÉ ILLIMITÉE

- 16 millions d'enregistrements et 999 index par fichier, nombre de champs illimité par enregistrement, nombre de fichiers illimité.
- Enormes capacités d'exploitation des fichiers et des index.
- Gamme complète de formats numériques.
- Calendrier 1-9999 A.C. Grand choix de formats de date.
- Options automatiques de titre, de date et de numérotation des pages.
- Les états peuvent comprendre compteurs, moyenne, sous-totaux et totaux.
- Largeur maximale des états 255 colonnes.
- Protection par mot de passe sur 3 niveaux.

PUISSANT

- Tri et édition de n'importe quelle combinaison de champs.
- Champs de type "formule" pour effectuer des calculs automatiquement et instantanément.
- dexation sur n'importe quel champ en vue d'une recherche immédiate des enregistrements ou d'une édition personnalisée.
- Flexibilité parfaite de sélection et d'exploitation de données provenant de fichiers différents.
- Sélection et utilisation d'une catégorie particulière d'enregistrement.
- Vérification et validation des données à la saisie pour assurer des enregistrements exacts.

BASIC GFA

un logiciel pour ATARI ST^F

Le Basic GfA est le langage de programmation le plus simple pour exploiter à fond toutes les possibilités de votre ATARI STF (520 et 1040). Il met à votre disposition plus de 200 commandes très puissantes pour réaliser des logiciels en BASIC, avec entre autres, la gestion de la souris, des menus, des zones d'alertes, etc.

Très rapide et souple d'emploi, il vous permettra de créer rapidement des applications de qualité professionnelle et d'une finition parfaite.

LE BASIC GfA tire profit des principes de la programmation structurée grâce à son jeu d'instructions très complet (WHILE..WEND, REPEAT..UNTIL, DO..LOOP, etc.).

Quelques caractéristiques :

- Très rapide (0.4 secondes pour boucle à vide FOR..NEXT exécutée 10 000 fois).
- Interfacé avec GEM (création des menus, gestion de la souris, des fenêtres, ...).
- Programmation structurée (procédures, variables locales...).
- Interpréteur compact (56 Koctets) qui laisse une grande place mémoire disponible même sur un 520 STF.
- Interpréteur RUN-ONLY librement copiable permettant l'exécution de programmes sans passer par l'éditeur (pour une commercialisation éventuelle).
- Commandes de graphismes (remplissage, ellipses, ...).
- Editeur très souple avec indentation automatique.
- Fonctionne dans les trois résolutions (320 × 200, 640 × 200 et 640 × 400).
- Gestion des périphériques.
- Gestion du système (interface avec langage C...).
- Commandes sonores.
- Précision de 11 chiffres significatifs.
- Commandes de gestion de fichiers.
- Commandes-systèmes (Call, Timer, Xbios, Void...).
- Commandes en mode superviseur (accès aux plages réservées).
- Etc.

Que vous vouliez tirer profit de votre ATARI STF ou simplement réaliser vos propres programmes, le BASIC GfA est le langage qu'il vous faut.

Système requis :

- Un ATARI 520 STF ou 1040 STF.
- Un moniteur monochrome ou couleur.
- Une imprimante (facultatif).

ISBN : 2-86899-071-1

Réf. : ST 012